ICADE BUSINESS SCHOOL

OPTIMIZACIÓN DE CARTERAS MEDIANTE APRENDIZAJE POR REFUERZO PROFUNDO

Clave: 201606171

MADRID | Junio 2022

Resumen

La gestión de portfolios tiene como objetivo asignar recursos a una cartera con el fin de optimizar los beneficios esperados por unidad de riesgo asumido. La dificultad de establecer un modelo que se ajuste a los cambios y la volatilidad del mercado es la piedra angular para el éxito de cualquier estrategia de inversión. Son precisamente la incertidumbre y la presunta ineficiencia de los mercados las que han llevado a la aparición de distintos enfoques por parte de los gestores de carteras. Para superar la debilidad e irracionalidad de las decisiones humanas se han diseñado numerosas estrategias de inversión a lo largo de la historia. En este sentido, el vertiginoso desarrollo de la tecnología aplicada a los mercados financieros se ha producido en un doble sentido. Por una parte, los economistas han flexibilizado su percepción de la eficacia del mercado aceptando la posibilidad de cierta capacidad predictiva de los rendimientos del mismo por medio de algoritmos no lineales. Por otra parte, se ha experimentado una transformación sustancial desde el punto de vista de las herramientas empleadas, adaptando los avances registrados en otras disciplinas. En este sentido, el empleo de sistemas inteligentes de optimización desempeña un papel fundamental.

El modo natural de aprendizaje del ser humano ha servido previamente como fuente de inspiración para otros procedimientos de algoritmos computacionales. Este ha sido, también, el caso reciente del aprendizaje por refuerzo, cuyos fundamentos tienen aplicaciones muy prometedoras en el ámbito bursátil. Este algoritmo trata de emular el comportamiento de cualquier ser vivo mediante técnicas empleadas de modo ascentral en la naturaleza como la prueba y el error. En consecuencia, se pueden extrapolar dichos mecanismos al entorno financiero y simular las recompensas que los agentes van adquiriendo tras la compraventa de valores en el mercado bursátil. Este tipo de aplicación es precisamente lo que se desarrolla en este trabajo.

Abstract

Portfolio management aims to allocate resources to a portfolio in order to optimize expected returns per unit of risk assumed. The difficulty of establishing a model that adjusts to market changes and volatility is the cornerstone for the success of any investment strategy. It is precisely the uncertainty and presumed inefficiency of the markets that have led to the emergence of different approaches by portfolio managers. To overcome the weakness and irrationality of human decisions, numerous investment strategies have been designed throughout history. In this regard, the dizzying development of technology applied to financial markets has been twofold. On the one hand, economists have relaxed their perception of market efficiency by accepting the possibility of a certain predictive capacity of market returns by means of non-linear algorithms. On the other hand, there has been a substantial transformation in terms of the tools used, adapting the advances made in other disciplines. In this sense, the use of intelligent optimization systems plays a fundamental role.

The natural behavioral mode of human learning has served as a source of inspiration for other algorithmic procedures. This has also recently been the case for reinforcement learning, the fundamentals of which have very promising applications in the stock market domain and have led to the development of reinforcement learning, the fruit of which seems to be quite promising in the stock market domain. Reinforcement learning attempts to emulate the learning behavior of any living being by means of techniques used ancestrally in nature, such as trial and error. Consequently, these mechanisms can be extrapolated to the financial environment and simulate the rewards that agents acquire after buying and selling securities in the stock market. This type of application is precisely what is developed in this work.

Índice general

Índice de figuras

Índice de tablas

Acrónimos

DRL	Deep Reinforcement Learning
RL	Reinforcement Learning
MDP	Proceso de decisión de Markov
DQN	Deep Q-Networks
DRQN	Deep Recurrrent Q-Network
DDPG	Deep Deterministic Policy Gradient
BH	Buy & Hold
GR	Gated Recurrent Units
SR	Ratio de Sortino
RNN	Red neuronal recurrente
CNN	Red neuronal convolucional
LSTM	Long short-term memory
PPO	Proximal policy optimization
A2C	Advantage Actor Critic
RCNN	Red neuronal convolucional recurrente
GDQN	Gated Deterministic Policy Gradient
DSR	Ratio de Sharpe diferencial
PPO	Proximal policy optimization
EIEE	Ensemble of Identical Independent Evaluators
KL	Kullback-Leibler
TRPO	Trust region proximal policy optimization
SVM	Support Vector Machine

NFQ	Neural Fitted Q-Learning
NFQCA	Monte Carlo Tree Search
CRISP-DM	Cross Industry Standard Process for Data Mining
SAC	Soft Actor Critic
TD3	Twin Delayed DDPG
MPT	Modern Portoflio Theory

Capítulo 1

Introducción

1.1. Motivación

La correcta predicción de los mercados financieros es una antigua quimera perseguida desde sus inicios por los inversores. El diseño de estrategias de inversión eficientes ha sido el epicentro de controversias del mundo financiero en las últimas décadas. Entre ellos, la optimización dinámica de la cartera es uno de los problemas más frecuentes a los que se enfrentan los profesionales del sector de la inversión. Sus tres paradigmas principales son: la Media-Varianza, el crierio de Kelly y la paridad de riesgo. Bajo el criterio de Media-Varianza, se persiguen inversiones en la llamada frontera eficiente, que se define como el conjunto de inversiones que producen mayor rendimiento posible para cualquier nivel de riesgo. Por su parte, el fundamento detrás del criterio de Kelly busca maximizar la tasa de crecimiento geométrico esperado de una determinada cartera. Por último, la paridad de riesgo buscar igualar el riesgo de los distintos componentes de la cartera mediante ponderaciones inversamente proporcionales a la volatilidad de la rentabilidad del componente en cuestión. En realidad, el criterio de Kelly es un caso especial de la Media-Varianza y la paridad de riesgo se convierte también en ella bajo unas condiciones específicas sobre la correlación de la rentabilidad y el ratio de Sharpe. Estos dos hechos implican que el problema de la optimización de la cartera podría tener alguna solución de carácter universal.

El problema se caracteriza por ser un proceso dinámico e intertemporal de determinación de las ponderaciones óptimas de la cartera que maximizan el rendimiento esperado de la misma para un determinado nivel de riesgo. La incertidumbre sobre los estados futuros del mercado, es decir, la dificultad de predecir

secuencialmente las inversiones con una precisión suficiente, hace que sea un problema de control estocástico óptimo en espacios de estado y acción continuos y por lo tanto, modelable bajo las premisas del aprendizaje por refuerzo.

El aprendizaje por refuerzo se ocupa de diseñar *agentes* que interactúan con un *entorno* y aprenden por sí mismos a recopilar el máximo número de recompensas por el método de ensayo y error de forma sistemática. Un entorno puede ser un juego de ajedrez o las carreras, o incluso resolver una tarea como solucionar un laberinto. El agente es el *bot* que realiza la actividad y recibe recompensas al interactuar con el entorno. Por lo tanto, este aprende a realizar las acciones necesarias para maximizar las recompensas que recibe de su ambiente. Un entorno se considera resuelto o satisfactorio, si el agente acumula algún umbral de recompensa predefinido (e.g., consigue salir del laberinto en menos de X movimientos). De la misma manera que un robot aprende mediante ensayo y error a salir de un laberinto o jugar al ajedrez, se puede enseñar a un agente a tomar decisiones óptimas para maximizar el valor de una cartera en base a datos históricos. Extrapolando los conceptos al ámbito financiero, las acciones son las decisiones del agente sobre qué pesos asignar a cada activo de su cartera, las recompensas son los beneficios derivados de la compraventa de dichos valores y el entorno sería el mercado financiero, un sistema complejo, no estacionario, altamente ruidoso y dominado por la no linealidad.

El objetivo fundamental de este Trabajo Fin de Grado es diseñar un algoritmo que, inspirado en las nociones teóricas del aprendizaje por refuerzo, sea capaz de encontrar una solución óptima para el problema de la asignación de capital de una cartera de inversión.

1.2. Objetivos

Por todo lo mencionado anteriormente, el trabajo propuesto gira en torno a la consecución de los siguientes objetivos:

- Contrastar el éxito de estrategias de inversión que han simulado técnicas análogas a las presentadas para, así, conocer diferentes mecanismos empleados en el entorno financiero y detectar engranajes susceptibles de adaptación al modelo diseñado.
- Estudiar las bases teóricas que sustentan el desarrollo de algoritmos de apren-

dizaje por refuerzo y diseñar un modelo que sea capaz de automatizar las decisiones financieras.

- Mejorar la gestión de una cartera de valores mediante la optimización de los pesos empleando el método propuesto y comparar la eficiencia del mismo con respecto a los modelos clásicos más frecuentes del sector.

1.3. Metodología

A continuación, se presenta el procedimiento desarrollado para la consecución de cada uno de los objetivos descritos anteriormente. Mediante la revisión del estado del arte, se profundizará en las investigaciones que se han desarrollado en este entorno. Con ello, se podrá estudiar la viabilidad de los distintos avances alcanzados y tomarlos como semillas para poder generar los frutos deseados.

El diseño de estrategias de inversión bursátiles es el núcleo de los debates sobre gestión financiera financieras de las últimas décadas. Por ello, es necesario conocer los entresijos que se esconden tras la toma de decisiones económicas. Para ello, se estudiarán las bases que sustentan la optimización de los portfolios y las diferentes vías de proliferación que se han propagado en este entorno.

Una vez que se tenga un conocimiento exhaustivo del problema a resolver, se procederá con la implementación del modelo basado en las técnicas de aprendizaje por refuerzo. En primer lugar, se construirá una base de datos con el histórico de precios de las 30 acciones del índice Dow Jones, incluyendo la información de precios diarios e intradiarios, así como distintos indicadores técnicos a lo largo de 10 años. A continuación, se desarrollará un entorno que permita la simulación de diferentes estrategias de inversión con numerosos algoritmos de Reinforcement Learning (RL) e implementando un gran abanico de posibilidades para su gestión, entre ellas, diferentes métodos para inicializar los pesos, varias alternativas de funciones de recompensas y numerosos criterios para seleccionar las entradas del modelo. Posteriormente, se construirá un backtesting que permita comparar los diferentes modelos de RL y, una vez elegido el mejor de ellos, se enfrentará su rendimiento contra tres becnhmarcks usados habitualmente en el sector: la estrategia de comprar y mantener (Buy & Hold) un ETF del DJI, la estrategia de un portfolio equiponderado y los resultados bajo un criterio como el de Media-Varianza (Markowitz). Con esto, se tendrá una excelente representación de la validez del método propuesto en distintos escenarios y simulaciones propuestas.

1.4. Recursos

La automatización de las decisiones financieras se codificarán empleando *Python* como lenguaje de programación. Para la implementación del algoritmo de aprendizaje por refuerzo se tomará por referencia la librería *FinRL*. A parte de la herramienta mencionada, se dispondrá de un conjunto de datos obtenidos de *Yahoo Finance* que proporcionan la evolución temporal de los precios máximos, mínimos, de apertura y cierre, y el volumen de transacciones para los 30 componentes del índice Dow Jones durante los últimos 10 años.

1.5. Estructura de la memoria

El presente trabajo está configurado atendiendo al siguiente esquema. En el segundo capítulo, se pretende revisar el estado del arte en lo que se refiere a las investigaciones sobre la aplicación del aprendizaje por refuerzo en el mercado financiero en general, y luego enfatizando los estudios realizados para la optimización de portfolios en particular. En el tercer capítulo se expone la teoría fundamental para entender las conjeturas aplicadas en la construcción del algoritmo de aprendizaje por refuerzo. En el cuarto capítulo se explica cómo se han integrado dichos algoritmos para automatizar la asignación de capital de la cartera de forma práctica. Para finalizar en el quinto capítulo, se incorporan los resultados basados en experimentos que pretenden desafiar la validez del modelo presentado.

Capítulo 2

Estado del arte

La utilización de algoritmos para operar en el mercado financiero ha ido ganando presencia en la industria como consecuencia del desarrollo tecnológico. El trading algorítmico se ha convertido en un elemento básico del mercado financiero actual, puesto que la mayoría de las operaciones están completamente automatizadas. Los agentes de aprendizaje por refuerzo profundo, en adelante *DRL*, han demostrado ser una fuerza a tener en cuenta en muchos juegos complejos como el ajedrez y el Go [1]. En este sentido, se pueden interpretar las series de precios y los movimientos históricos del mercado de valores como un complejo entorno de información imperfecta en el que se trata de maximizar la rentabilidad y minimizar el riesgo de forma simultánea. Un *juego* en el que quizá el ensayo y error pueda llevar a desentrañar las soluciones óptimas. De este modo, el objetivo fundamental de esta sección es revisar los avances realizados en materia del aprendizaje por refuerzo en el subdominio de la Inteligencia Artificial aplicada a las finanzas, y de forma más concreta, situando el foco en las estrategias cuantitativas de baja frecuencia.

Muchos de los estudios revisados sólo disponen de pruebas de conceptos, con experimentos realizados en entornos poco realistas y sin ninguna aplicación directa en tiempo real. De forma general, aunque en la mayoría de los trabajos analizados se prueban mejoras estadísticas significativas en comparación con las estrategias de referencia establecidas, no se obtienen niveles de rentabilidad robustos ni sostenibles. Para realizar una revisión exhaustiva y sistemática de la literatura existente, es preciso enunciar los principales retos a os que se enfrenta el RL cuando se trata de operaciones financieras:

[1] AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

- La frecuencia de adquisición de los datos, así como la rapidez con la que estos datos llegan y el tiempo que el algoritmo tarda en incorporarlos al proceso de decisión para ejecutar una acción óptima.

- La ineficiencia del mercado. En el sentido en que los precios pueden estar afectados por *shocks* transitorios que no tienen que ver con su valoración.

- La búsqueda de un equilibrio en el dilema entre la exploración y la explotación, ya que para que un DRL tenga éxito suele ser necesaria una gran exploración de soluciones, pero sin embargo, esto no es factible en el mundo financiero ya que la exploración aleatoria generaría inevitablemente una gran cantidad de costes de transacción y por lo tanto, una pérdida del capital disponible.

- Por último, y más importante, de forma muy simple, sólo se pueden optimizar en base a sucesos que han ocurrido y de los que se disponen datos. Se puede diseñar que el algoritmo sepa que cuando se da X situación lo óptimo es tomar Y decisión. Pero si el algoritmo nunca se ha enfrentado a una situación X, entonces no puede aprender del pasado en un sentido estricto. Esto refleja el clásico problema *in-sample vs. out-of-sample*.

Como ya se ha comentado anteriormente, el RL trata de maximizar la recompensa esperada de un entorno que suele modelarse como un proceso de decisión de Markov (MDP). Es precisamente la optimización de la política que el agente debe tomar, lo que divide el paradigma del RL en dos vertientes fundamentales y esto a su vez en tres técnicas principales: aprendizaje de sólo crítico, de sólo actor y de actor-crítico. En primer lugar, se revisarán todos los avances realizados en materia de RL en el mercado financiero de forma general, dividiendo esta primera sección según las funciones que se empleen para aproximar la función de política. En segundo lugar, se situará el foco en los estudios realizados en el entorno de la gestión de una cartera, tanto para los métodos de RL basados en valores como en políticas. De este modo, se excluyen de esta sección el estudio de investigaciones de métodos basados en modelos para RL, ya que no existe una clara adaptación al respecto.

Previo a la revisión del estado del arte, conviene introducir un ejemplo clásico del RL como punto de partida. El problema del taxi es uno de los muchos entornos disponibles en OpenAI Gym [2]. Estos entornos se utilizan para desarrollar y comparar algoritmos de aprendizaje por refuerzo. El objetivo del taxi es recoger a los

[2]https://www.gymlibrary.ml/

pasajeros y dejarlos en su destino en el menor número de movimientos. El entorno está definido por una matriz de 5×5, donde cada celda es una posición en la que el taxi puede permanecer. Entonces, se tienen 4 coordenadas que representan las ubicaciones de recogida y entrega, que son (0,0), (0,4), (4,0), (4,3). Se denotan, respectivamente, como R,G,Y,B y se indexan su ubicación con 0,1,2,3. Por último, hay un pasajero que puede ser recogido o dejado, además de ser transportado (por lo que pasa tiempo en el taxi). En concreto, este pasajero quiere llegar al punto B. Dicho entorno se puede visualizar en la Figura 2.1.

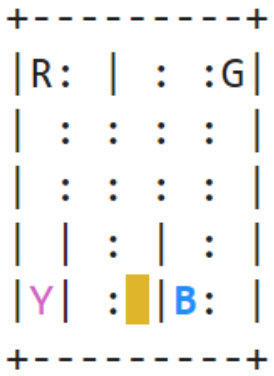


Figura 2.1. *Ejemplo del entorno del taxi en OpenAI Gym*

Como se puede observar, se dispone de un espacio de 5×5 con 4 ubicaciones, donde la letra azul representa la posición del pasajero actual, mientras que la morada es la ubicación de bajada. También se encuentra el taxi/agente en ese espacio, que es el rectángulo amarillo, así como algunas paredes, representadas por el símbolo '|'.

Para entender el mecanismo del RL es necesario introducir dos conceptos claves: los estados y las acciones. En primer lugar, se examinarán las acciones. Según el entorno descrito, el agente puede actuar de 6 maneras:

1. Bajando (sur)
2. Subiendo (norte)
3. Ir a la derecha (este)
4. Ir a la izquierda (oeste)

5. Recoger

6. Dejar el coche

En segundo lugar, se dispone de al menos 25 estados (matriz 5×5). Además, el taxi puede estar también en estado de recoger o dejar al pasajero (independientemente de que esté realmente allí), de ahí que aparezcan 4 estados adicionales. Por último, es necesario computar aquellos estados en los que el pasajero es realmente recogido, o simplemente transportado. De modo que en total se tienen $5 \times 5 \times 4 \times 5 = 500$ estados. Cada uno de ellos, que está representado por un vector de valores [fila del taxi, columna del taxi, índice del pasajero, índice del destino], está consecuentemente codificado con un valor entre 0 y 499.

El sistema de recompensa es la idea principal del aprendizaje por refuerzo: el agente es recompensado cada vez que actúa bien, de lo contrario es castigado con una recompensa negativa:

- Si el taxi recoge/deja correctamente al pasajero, es recompensado con +20 puntos.
- Si el taxi recoge/deja al pasajero de forma ilegal, se le castiga con -10 puntos. Por cada paso que no incluya los estados anteriores, pierde 1 punto.

De este modo, la terna del estado, acción y la función de recompensa son tres de los cinco elementos claves de cualquier algoritmo de RL. Por consiguiente, el objetivo del taxi es completar su función, dentro de los estados posibles, con el menor número de acciones para, así, lograr la máxima recompensa.

Una vez que se ha ilustrado la dinámica general de cualquier algoritmo de RL se prosigue con la revisión de la literatura existente.

2.1. Aplicación de las técnicas de aprendizaje por refuerzo al mercado financiero

2.1.1. Aprendizaje por refuerzo de sólo crítico

Previo a analizar los estudios de esta categoría, es importante destacar las principales limitaciones de las que adolece este enfoque. El método crítico, representado

principalmente por el DQN (*Deep Q-Networks*) y sus variaciones mejoradas, es el más publicado en esta área de investigación (Arulkumaran et al. (Arulkumaran y col., 2017), Li (Li, 2017), Murat et al. (Ozbayoglu y col., 2020) y Thakkar & Chaudhari (Thakkar y Chaudhari, 2021). Sin embargo, su principal limitación es que funciona de forma satisfactoria para representaciones de estados discretas, pero no tan bien para procesos continuos, como los precios de las acciones. De este modo, resulta crucial la manera de implementación de la función de recompensa ya que es un método muy sensible a las señales que recibe del entorno.

Existen numerosas investigaciones inspiradas en tratar de replicar el éxito de juegos como el Atari y el Póker. Entre ellas cabe destacar el avance realizado por Chen y Gao (Chen y Gao, 2019) en el que introducen una variante del sistema de aprendizaje DRQN, (*Deep Recurrent Q-Network*), que utiliza una red neuronal recurrente permitiendo procesar mejor las secuencias temporales de las series de precios. Los autores utilizan el conjunto de datos históricos de 19 años de los precios de cierre diarios del ETF S&P500 (SPY) para entrenar y probar el agente contra los puntos de referencia: una estrategia de comprar y mantener a largo plazo (usada por agentes humanos) y un operador DQN de acción aleatoria. De este modo, definen una representación de los estados mediante una secuencia de datos de precios de cierre ajustados a lo largo de una ventana de 20 días. Al tratarse de un enfoque puramente crítico, cualquier agente tiene un conjunto finito de acciones: comprar, vender o mantener y, su recompensa se calcula como la diferencia entre el precio de cierre ajustado del día siguiente y del día actual. Los resultados, representados en la Figura 2.2, prueban que el agente DRQN bate (in-sample) a los modelos de referencia alcanzando una rentabilidad anual de entorno al 23 %.

Sin embargo, existen claras limitaciones en el estudio de este documento. En primer lugar, sólo se considera una acción como referencia. Esto no es necesariamente erróneo, pero sí cuestiona la generalidad del enfoque propuesto a una cartera más amplia. En segundo lugar, la función de recompensa es muy simple ya que se centra únicamente en maximizar la rentabilidad del periodo sin tratar de reducir el riesgo. En tercer lugar, el espacio de acciones que el agente puede realizar es muy limitado. Por último los datos de los precios en bruto contienen demasiado ruido que altera significativamente los resultados y la reproducibilidad del modelo.

Existen estudios estrechamente relacionados con este, como el realizado por Chien Yi (Huang, 2018), en el que también se emplea un modelo DQRN como agente. Sin embargo, se pueden observar diferencias significativas. En primer lu-

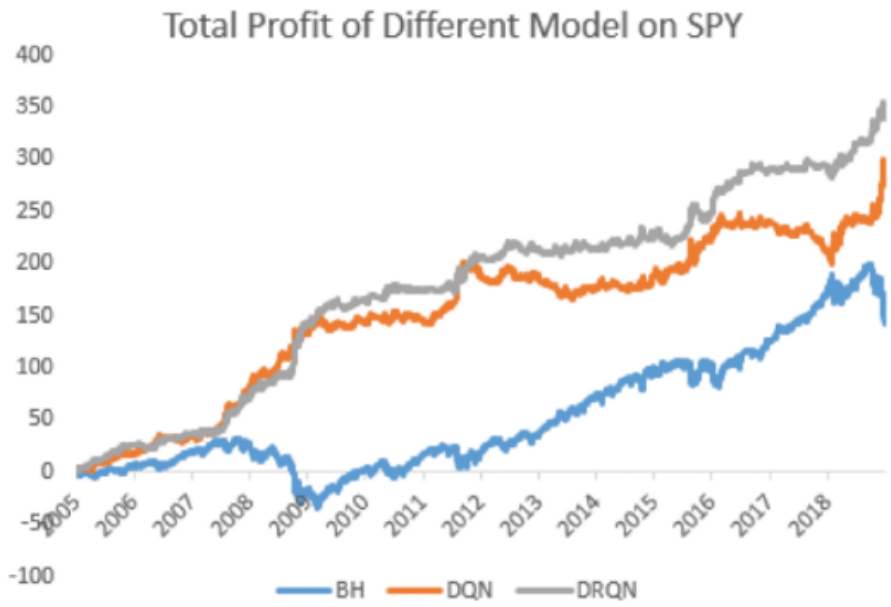


Figura 2.2. *Rentabilidad de BH, DQN Y DRQN en el conjunto de Test del SPY (Chen y Gao, 2019)*

gar, utilizan una combinación de 16 características en total, incluyendo el OHLCV (Open, High, Low, Close & Volume) diario e indicadores de precio. En segundo lugar, durante el entrenamiento se utiliza una memoria de repetición sustancialmente pequeña y se introduce una técnica para mitigar la necesidad de exploración aleatoria proporcionando señales de retroalimentación adicionales para todas las acciones del agente. Esto permite el uso de políticas que emplean el factor $\epsilon greedy$ para favorecer el equilibrio entre la exploración y explotación previamente descrito. El algoritmo se aplica en el mercado de divisas durante el periodo comprendido entre 2012 y 2017, a 12 pares de divisas, durante un marco temporal de 15 minutos y teniendo en cuenta los costes de transacción. Los resultados globales del mismo proporcionan una rentabilidad del 10 % anual, alcanzando algunos valores el 60 %, lo que supone un avance con respecto al artículo presentado.

Por otra parte, existen otras investigaciones como las realizados por Xing et al. (X. Wu y col., 2020), Nabipour et al. (Nabipour y col., 2020) y Soleymani et al. (Soleymani y Paquet, 2020) que comparan este enfoque con los dos restantes. En el propuesto por Xing et al. (X. Wu y col., 2020) se presentan las *Gated Recurrent Units (GRUs)* como un método de aprendizaje profundo para extraer

características para el sistema *Q-Learning (GDQN)*. En la Figura 2.3 se incluye un esquema del mismo. La principal motivación para introducir este factor es que los movimientos del mercado no pueden revelar los patrones o características que hay detrás de los estados dinámicos del mercado. Por ello, para hacer frente a esta limitación, los autores presentan una arquitectura con una puerta de actualización y otra de reinicio para la red neuronal. La puerta de reinicio actúa como un filtro para la información bursátil anterior, manteniendo sólo una parte de ella, y la puerta de actualización decide si el estado oculto se actualiza o no con la información actual. Además, otra contribución de este trabajo es el uso de una función de recompensa que considera mejor el riesgo, empleando el ratio de Sortino (SR) (Mohan y col., 2016) para ello. Entre los trabajos revisados hasta el momento, este es el que dispone la configuración más prometedora. Sin embargo, sigue habiendo una clara limitación en las acciones que el agente puede realizar (sólo 1 acción por transacción).

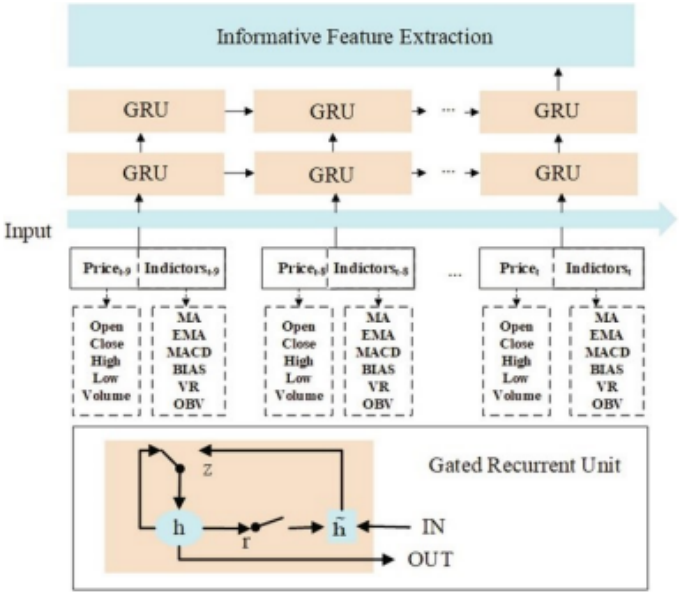


Figura 2.3. *Arquitectura de una neurona GRU (X. Wu y col., 2020)*

2.1.2. Aprendizaje por refuerzo de sólo actor

Bajo esta nueva perspectiva, como la política se aprende directamente, el espacio de acciones es más generalizable y, por lo tanto, se pueden definir acciones de

forma continua. La principal ventaja derivada de este factor es que se aprende de una asignación directa, es decir, de lo que hay que hacer en un estado en concreto. Sin embargo, esto conlleva por lo general un mayor tiempo de entrenamiento para el aprendizaje de las políticas óptimas (Fischer, 2018).

Entre la literatura disponible destaca el artículo realizado por Yue et al (Deng y col., 2017) en el que tratan de responder a la pregunta de si un algoritmo de aprendizaje por refuerzo puede batir a seres humanos con gran experiencia en el sector financiero. El enfoque propuesto es muy novedoso ya que remodelan la estructura de la tradicional red neuronal recurrente (RNN) para la detección simultánea del entorno y la toma de decisiones bajo un entorno online. El método emplea el RL para la extracción de características combinando al mismo tiempo estas nociones con conceptos de aprendizaje difuso (Pal y Bezdek, 1984) [3] para reducir la incertidumbre de los datos de entrada. Asimismo, introducen una variante de la retropropagación en el tiempo para lidiar de forma efectiva con el problema del desvanecimiento del gradiente en estructuras complejas. La parte de los experimentos es sólida desde el punto de vista de la comparación de las estrategias establecidas en la literatura y los enfoques propuestos. No obstante, carece de simulaciones suficientes en una cartera diversa, ya que, sólo se eligen 3 contratos (dos de materia prima y un futuro del índice bursátil IF).

Las comparaciones de los contratos de índices bursátiles y de futuros de materias primas muestran que el sistema de aprendizaje por refuerzo directo (DRR) y su extensión difusa (FDDR) generan significativamente más beneficios totales que sus respectivos benchmarks. Un grave inconveniente de este estudio es la inconsistencia de la respuesta con respecto a la formulación de la cuestión inicial, puesto que, en ninguno de los experimentos se compara con el rendimiento de un ser humano experto en la materia. Además, no se especifican si los valores del ratio de Sharpe, en torno a 9, están anualizados. Los resultados de la investigación se incluyen en la Figura 2.4.

Por otra parte, cabe destacar la investigación llevada a cabo por Wu et al (J. Wu y col., 2019). Sus principales contribuciones son la realización de un análisis exhaustivo de la utilización de redes LSTM frente a redes completamente profundas, así como el impacto de algunas combinaciones de los indicadores técnicos en el mercado chino con datos diarios.

[3] La lógica difusa, es una lógica que permite llegar a conclusiones "razonadas" a partir de información ambigua o imprecisa

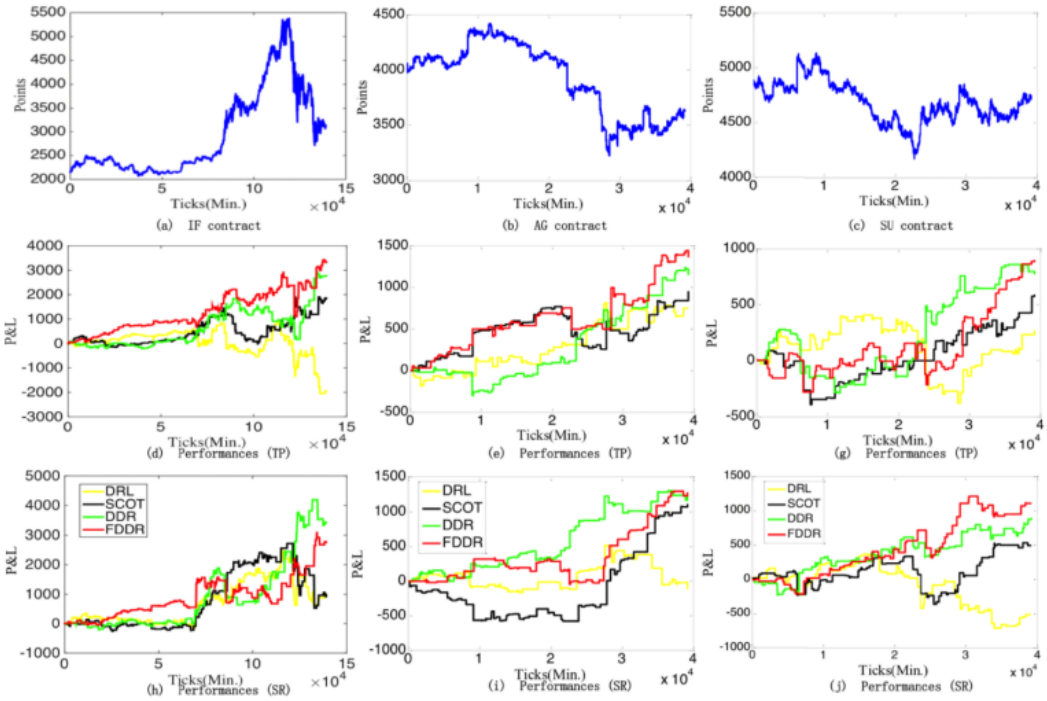


Figura 2.4. *Comportamiento del mercado, rentabilidad de diferentes sistemas de negociación y ratio de Sharpe móvil (Pal y Bezdek, 1984)*

2.1.3. Aprendizaje por refuerzo de actor-crítico

La principal ventaja de este tipo de algoritmos es que se ajustan mejor a entornos complejos, ya que, los datos que se generan durante el entrenamiento se aprenden de la política actual permitiendo así, una mejor aproximación de las distribuciones de los datos a medida que se actualizan las recompensas que el agente va adquiriendo. Sin embargo, a pesar de ser uno de los enfoques más exitosos de la literatura parece ser un método aún inexplorado.

Entre las publicaciones existentes cabe destacar el estudio realizado por Hongyang et al. (Yang y col., 2020) en el que introducen un método de ensamblaje combinando tres algoritmos de RL de última generación: PPO, (*Proximal policy optimization*), A2C, (*Advantage Actor Critic*), DDPG, (*Deep Deterministic Policy Gradient*). En la Figura 2.5 se muestra un esquema del sistema descrito. Este enfoque otorga una mayor robustez a la estrategia en conjunto, siendo capaz también de adaptarse mejor a las diferentes situaciones del mercado, ya que determinados

agentes pueden presentar una mayor sensibilidad frente a diferentes tendencias y volatilidades del entorno financiero. Asimismo, se observa una notoria mejoría en cuanto a la definición de la gestión de la cartera de acciones ya que la elección del espacio de acciones que el agente puede tomar por acción explora un espacio más extenso. El método propuesto supera al índice Dow Jones en el periodo analizado, tal y como se puede observar en la Figura 2.6.

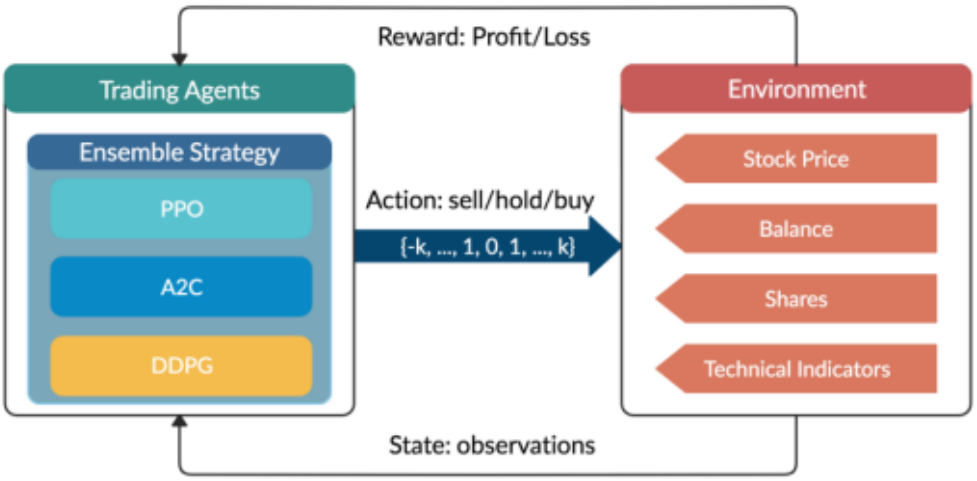


Figura 2.5. *Visión de conjunto de la estrategia DRL (Yang y col., 2020)*

Un artículo muy relacionado con el anterior es el presentado por Zhuoran et al. (Xiong y col., 2018) en el que estudian el algoritmo DDPG bajo las mismas condiciones de partida y métricas, pero sin la evaluación de indicadores técnicos. Analizan el rendimiento del algoritmo durante los años comprendidos entre 2015 y 2018 y obtienen una rentabilidad anualizada del 25,87 % en el índice Dow Jones, más que el 15,93 % obtenido mediante el método de mínima varianza y el 16,4 % de la media industrial. Las comparaciones sobre los ratios de Sharpe también son favorables (1,79 frente a 1,45 y 1,27, respectivamente). Por lo tanto, parece que se trata también de un método más robusto en relación con la aversión al riesgo del inversor.

Por otra parte, resulta interesante el enfoque presentado por Akhil et al. (Azhikodan y col., 2019) en el que combinan el RL con el análisis de sentimiento. Utilizando este factor, tratan de predecir los movimientos futuros de los mercados a

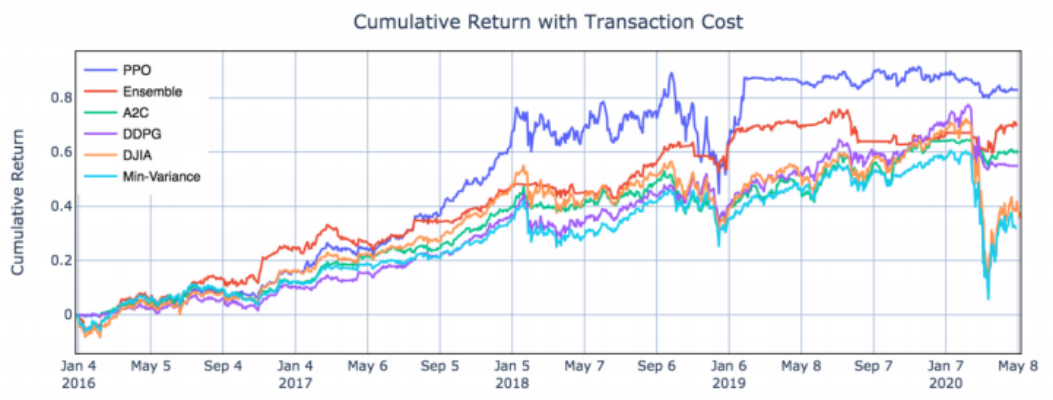


Figura 2.6. *Rentabilidad acumulada de diferentes estrategias sobre el índice de 30 acciones de EE.UU. Valor inicial de la cartera: 1 $M. (Yang y col., 2020)*

través de noticias financieras empleando una red neuronal convolucional recurrente (RCNN) para la clasificación de las mismas. De este modo, la RCNN predice si el precio subirá o no en los próximos días. Posteriormente, dicho bit binario se fusiona con el estado actual del agente del algoritmo de RL. La parte supervisada del método se entrenó con un corpus de 96 mil noticias financieras de cerca de 3500 compañías diferentes, resultando en un modelo con una tasa de acierto del 96,88 % evaluada en el conjunto de test. Aunque los experimentos no muestran un incremento abrumador de los beneficios, tal y como se puede observar en la Figura 2.7, sí que se pone de manifiesto que el agente aprende unas estrategias básicas: comprar y vender continuamente y mantener cuando hay una disminución del precio.

Por último, merece la pena discutir las investigaciones recogidas en dos artículos ya mencionados anteriormente. El primero de ellos, el estudio realizado por Xing et al. (X. Wu y col., 2020) en el que los autores presentan GDPG como una estrategia actor-crítica que combina la Q-Network del sistema GDQN, con una red para la función de política. En la Tabla 2.1 se demuestra que aplicando este método se logran rendimientos ajustados al riesgo más estables. Como benchmark emplean el sistema de trading de la Tortuga, introducido por Richard Dennis,como estrategia para diseñar estrategias de inversión que sigan tendencias.

El segundo de ellos, realizado por Zhang et al. (Zhang y col., 2020), compara directamente el algoritmo A2C con los enfoques restantes, representado en la Figura 2.8 los resultados experimentales. A la vista de los mismos, parece que el

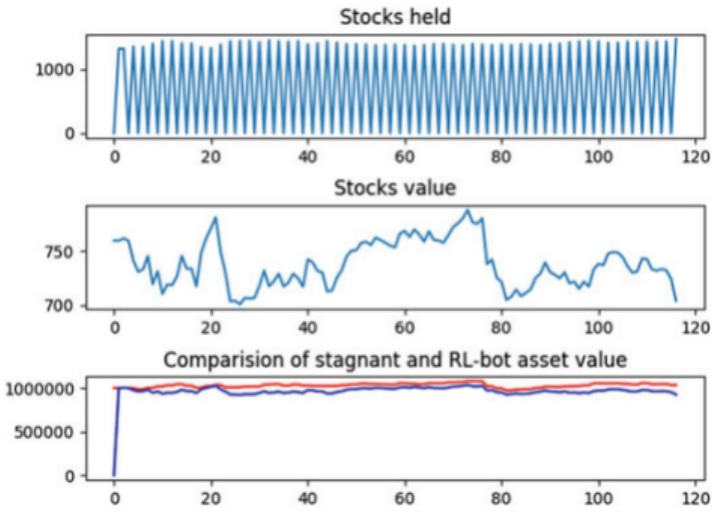


Figura 2.7. *Entrenamiento durante 5 meses con acciones de NASDAQ-GOOGL. Presupuesto inicial: 1 millón de dólares. La línea roja indica el patrimonio del agente, y la línea azul el valor de las acciones estancadas (Azhikodan y col., 2019)*

	GDQN		GDPG		Turtle	
Symbol	SR	R(%)	SR	R(%)	SR	R(%)
AAPL	1,02	77,7	1,30	82,0	1,49	69,5
GE	−0,13	−10,8	−0,22	−6,39	−0,64	−17,0
AXP	0,39	20,0	0,51	24,3	0,67	25,6
CSCO	0,31	20,6	0,57	13,6	0,12	−1,41
IBM	0,07	4,63	0,05	2,55	−0,29	−11,7

Tabla 2.1. *Comparación entre GDQN, GDPG y la estrategia de Tortuga en algunos valores estadounidenses durante un periodo de 3 años (2016-2019) (X. Wu y col., 2020)*

A2C queda en segundo lugar en cuanto a la rentabilidad de la cartera. Los autores justifican esto argumentando que dicho algoritmo genera mayores rotaciones y por ello, la rentabilidad es menor por rotación. Los experimentos también ponen de manifiesto que el A2C puede aprovechar los grandes movimientos del mercado sin cambiar de posición y que lidia de forma efectiva con los entornos financieros que tienen una reversión a la media más proclive.

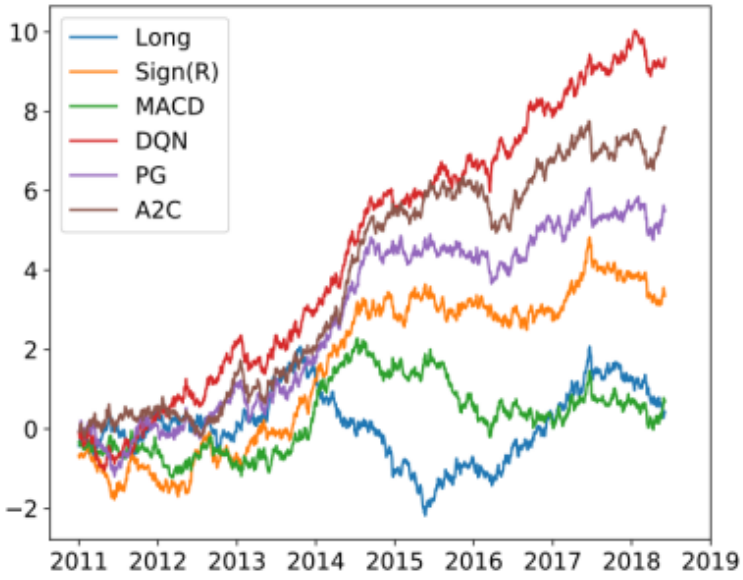


Figura 2.8. *Rendimiento acumulado de las operaciones. Por orden: materias primas, índice de renta variable y renta fija, FX y la cartera de utilizar todos los contratos (Zhang y col., 2020)*

2.2. Aplicación de las técnicas de aprendizaje por refuerzo a la asignación de capital en un portfolio

La gestión de carteras es uno de los problemas más frecuentes en el sector financiero. Sin embargo, no se reconoce de forma general que tanto el criterio de Kelly como la paridad de riesgo colapsan en el contexto media-varianza bajo algunas condiciones, lo que implica que podría existir una solución potencial que casara de forma universal. De hecho, el proceso de cálculo secuencial de las ponderaciones óptimas de los componentes que maximizan una cartera se puede reformular como un Proceso de Decisión de Markov (MDP) en tiempo discreto, y por lo tanto, como un control óptimo estocástico en el que el sistema que se controla es una cartera formada por múltiples componentes de inversión y el control son las ponderaciones óptimas de las mismas. En consecuencia, el problema se puede resolver aplicando la técnica de RL sin conocer la dinámica específica de los componentes. Para revisar el estado del arte de la cuestión se subdivide la sección en los métodos basados en valores y en políticas.

2.2.1. Métodos de RL basados en valores

En los modelos de RL basados en valores, como *Q-Learning*, la acción del agente no se determina encontrando la política óptima π directamente, sino obteniendo la función de valor estado-acción óptima, $Q(s, a)$. Esta se suele aproximar con una red neuronal debido a su capacidad de aproximación universal. El método basado en Q-Learning pretende aproximar directamente la función de valor óptima y los parámetros en cada época se aprenden de forma iterativa minimizando la siguiente función de coste:

$$L_i\left(\boldsymbol{\theta}_i\right) = \mathrm{E}\left[r + \gamma \max_{a'} Q\left(s', a'; \boldsymbol{\theta}_{i-1}\right) - Q\left(s, a; \boldsymbol{\theta}_i\right)\right]^2 \tag{2.1}$$

Du et al. (Du y Zhai, s.f.) usaron Q-Learning, sin emplear una red neuronal, para optimizar el peso de la liquidez en la cartera considerando los costes de transacción y con un factor de rebalanceo constante. Se construyeron tres funciones de valor estado-acción utilizando diferentes métricas: rentabilidad acumulada, ratio de Sharpe y el ratio de Sharpe diferencial (DSR). El DSR puede considerarse como la utilidad marginal en términos de la cantidad de riesgo que el inversor está dispuesto a soportar por un incremento unitario del ratio de Sharpe. Los autores observan que el rendimiento de la cartera varía significativamente según el tipo de función valor empleada y demuestran que la cartera que emplea la función DSR obtiene la mejor rentabilidad. El hecho de que el rendimiento de la cartera presente tanta variabilidad se atribuye a que el método de Q-Learning sufre gran inestabilidad con conjunto de datos ruidosos para la selección de la política óptima y por lo tanto, concluyen que los métodos de RL basados en políticas son más estables y eficiente que los basados en valores.

Jin & El-Saawy (Jin y El-Saawy, 2016) optimizan una cartera de dos acciones (una de β alta y otra de β baja) mediante el uso de Q-Learning con una red neuronal que aproxima la función valor estado-acción de forma similar a la DQN. Las características de entrada son los precios históricos de las acciones, el número de acciones en circulación, el valor total de la cartera y la cantidad de efectivo disponible. De este modo, la acción del agente es decidir el porcentaje de la cartera en la que se venden acciones de baja beta y se compran las de alta beta, cuyo espacio de acciones se discretiza en siete porcentajes ($\pm 25\,\%, \pm 10\,\%, \pm 5\,\%, 0\,\%$). La función de recompensa es la rentabilidad de la cartera o el ratio de Sharpe, ambos penalizados por la volatilidad de la cartera. El agente se entrena utilizando tanto una estrategia de exploración $\epsilon greedy$ como un búfer de repetición de experiencias similar al del DQN. El resultado de la prueba fuera de la muestra advierte que el

rendimiento de la cartera varía significativamente dependiendo de la métrica de rendimiento, la duración del historial de precios de las acciones y la penalización por volatilidad.

Weijs (Weijs, 2018) aborda el problema de la optimización de la cartera empleando también Q-Learning con una red neuronal que aproxima la función valor estado-acción con los rendimientos de los activos como una característica de entrada sin discretización y sin ninguna suposición sobre la distribución de los rendimientos. El autor construye una cartera compuesta por activos tanto de renta fija, a corto y largo plazo, como renta variable. Se afirma que el método Q-Learning logra un rendimiento comparable con un inversor neutral al riesgo, siendo la rotación de la cartera menor como resultado de la estrategia conservadora aprendida.

2.2.2. Métodos de RL basados en políticas

Algoritmos de gradiente de políticas deterministas profundas

En los métodos sin modelo de RL basados en políticas, la propia política se parametriza directamente con θ para representar la política óptima π. Para ello, el rendimiento objetivo se puede reescribir utilizando la densidad de probabilidad condicional $\pi\theta(a|s)$ como:

$$\begin{aligned} J\left(\pi_\theta\right) &= \int_S \rho^\pi(s) \int_A \pi_\theta(a \mid s) r(s,a) da ds \\ &= \mathrm{E}_{s\sim\rho^*, a\sim\pi_\theta}[r(s,a)] \end{aligned} \tag{2.2}$$

donde S es el espacio, A es la acción y ρ^π:

$$\rho^\pi\left(s'\right) = \int_S \sum_{t=1}^{\infty} \gamma^t p_0(s) p\left(s \rightarrow s', t, \pi\right) ds \tag{2.3}$$

Los parámetros θ se actualizan aplicando la técnica del descenso de gradiente estocástico según la siguiente fórmula:

$$\begin{aligned} \nabla_\theta J\left(\pi_\theta\right) &= \int_S \rho^\pi(s) \int_A \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s,a) da ds \\ &= \mathrm{E}_{s\sim\rho^2, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a)\right] \end{aligned} \tag{2.4}$$

La política por gradiente (PG) representa la política parametrizada según una distribución de probabilidad $\pi_\theta(a \mid s) = P(a \mid s; \boldsymbol{\theta})$ que selecciona la acción a en

el estado s según el vector de parámetros θ. La política determinista profunda por descenso (DDPG) combina DPG y DQN para largos dominios continuos en los que el actor aprende usando la ecuación de Bellmman del método de Q-Learning. Uno de los retos que plantea el uso de redes neuronales para RL es que la mayoría de los algoritmos de optimización suponen muestras i.i.d., lo que ya no se cumple cuando las muestras se generan explorando secuencialmente en un entorno. Tanto el DQN como el DDPG resuelven este problema utilizando un búfer de repetición de la experiencia. En cada paso de tiempo, el actor y el crítico se actualizan mediante el muestreo de un mini bacth de transiciones no correlacionadas del búfer.

Jian et al. (Jiang y col., 2017) presentan una solución del tipo DDPG para el problema de optimización de la cartera. Utilizan características de entrada que consisten en el OHLC de los precios y entrenan al agente con una función de política aproximada por una red neuronal profunda, llamada *Ensemble of Identical Independent Evaluators (EIIE)* (Figura 2.10), para calcular directamente el conjunto de pesos de la cartera. El DDPG sigue el gradiente de la función de valor estado-acción, que en su caso es simplemente el rendimiento muestral ajustado rt/tf obtenido inmediatamente después de tomar una acción, donde rt es la rentabilidad logarítmica de la cartera en el momento t, y tf es la duración del período completo de gestión de la cartera. A continuación, determina una acción dentro de un espacio continuo restringido basándose en las puntuaciones de votación de la última capa *softmax* de la red neuronal. El método empleado no tiene en cuenta la volatilidad de la cartera y acaba concentrando el peso total en un número muy reducido de activos, siendo además estas ponderaciones muy fluctuantes a lo largo del tiempo.

Asimismo, Guo et al. (Guo y col., 2018) utilizan una red neuronal convolucional (CNN) para aproximar directamente la función de política óptima, donde el objetivo del agente es maximizar el rendimiento logarítmico esperado de una cartera de 100 acciones seleccionadas al azar. En lugar de estimar las distribuciones de densidad de los rendimientos, aproximan las medias y las matrices de covarianza de las mismas, reduciendo la complejidad global del problema. Dicho procedimiento se realiza mediante la comparación de patrones empleando el coeficiente de correlación de Pearson entre los rendimientos de los precios actuales y los rendimientos históricos, y eligiendo el período de tiempo en el que el coeficiente es más alto. La red se entrena utilizando una repetición de la experiencia con un muestreo según la distribución de Poisson para enfatizar las experiencias más recientes. Sus pruebas retrospectivas aplicadas al conjunto de test muestran que

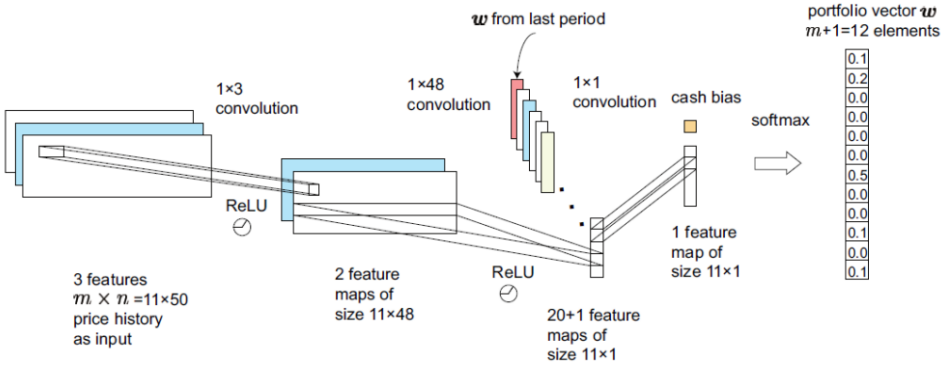


Figura 2.9. *Red neuronal convolucional (CNN) versióne la topología Ensemble of Identical Independent Evaluators (EIIE) Jiang y col., 2017*

el enfoque empleado supera los puntos de referencia asignados.

Algoritmos de aproximación de la política óptima

Aproximar la política óptima π directamente con una red neuronal con muchos parámetros es difícil y, a menudo, puede caer en óptimos locales debido principalmente a la inestabilidad y la ineficiencia de la muestra. Una solución frente a este problema es la optimización de la política empleando el algoritmo PPO. Este es una extensión del algoritmo de optimización de la política de la región de confianza (TRPO). Por lo tanto, para entender el funcionamiento del primero, es necesario previamente discutir las bases fundamentales del TRPO. La idea que subyace bajo este algoritmo es restringir cada actualización del gradiente de la política medida por la divergencia de Kullback-Leibler (KL) de dos distribuciones de probabilidad continuas, formulado matemáticamente mediante la siguiente ecuación:

$$D_{KL}(P,Q) = \mathrm{E}\left[\log \frac{p(x)}{q(x)}\right] = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx \tag{2.5}$$

Siendo $\eta(\boldsymbol{\theta}) = J(\pi_\theta)$ el rendimiento de una política estocástica parametrizada, $\pi\theta$, su función de límite inferior $M(\theta)$ puede definirse como:

$$\begin{aligned} \eta(\boldsymbol{\theta}) \geq M(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) - C \cdot KL \\ = \mathrm{E}\left[\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a \mid s)} \hat{\mathrm{A}}\right] - \frac{4\varepsilon\gamma}{(1-\gamma)^2} \cdot \max_s D_{KL}\left(\pi_{\boldsymbol{\theta}}(\cdot \mid s), \pi_{\boldsymbol{\theta}_{oud}}(\cdot \mid s)\right) \end{aligned} \tag{2.6}$$

Donde $L(\theta)$ es la función de ventaja esperada para la política actual calibrada por la relación de probabilidad de la misma con respecto al instante anterior. El uso de la función de ventaja en lugar de la recompensa esperada J reduce la varianza de la estimación. La Figura 2.10 muestra la relación entre ν, M, L, y, θ.

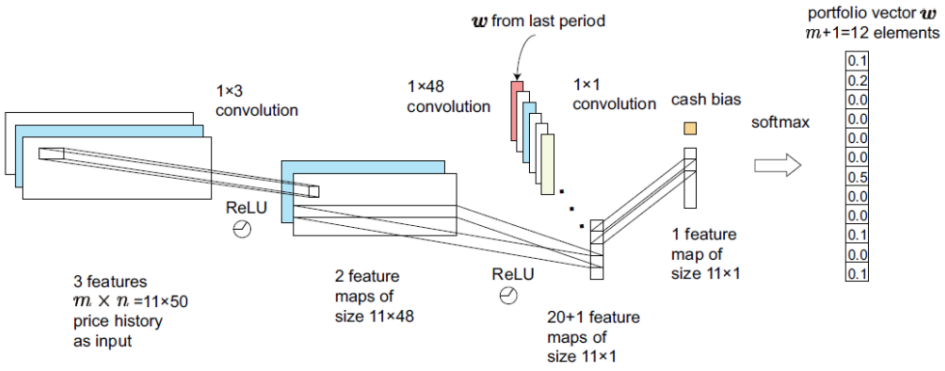


Figura 2.10. *Red neuronal convolucional (CNN) versióne la topología Ensemble of Identical Independent Evaluators (EIIE) (Jiang y col., 2017)*

Liang et al. (Liang y col., 2018) experimentan con el DPG, DDPG y el PPO construyendo carteras compuestas por cinco acciones chinas elegidas al azar y optimizaron periódicamente la cartera utilizando cada uno de los métodos de RL con una función de recompensa ajustada al riesgo y penalizada por la volatilidad media de los precios de los valores que la componen. Las entradas del modelo son los precios máximos y de cierre con un ruido aleatorio para implementar el aprendizaje adversario. Su DDPG también utilizan el esquema de *EIIE* introducido en (Jiang y col., 2017) pero utilizando una red residual profunda (ResNet) en lugar de la CNN para resolver el problema del desvanecimiento y explotación del gradiente, según se aumenta la profundidad de las redes. Sólo el DPG con el aprendizaje adversarial fue el único método de los tres propuestos que demuestra ser superior a las comparaciones bases.

En este capítulo se ha completado una revisión exhaustiva de los métodos de RL basado en valores y en políticas. La mayor desventaja de los primeros, como el Q-Learning, es la maldición de la dimensionalidad de Bellman que surge de los grandes espacios de estado y acción, lo que dificulta al agente la exploración eficiente. Du et al. (Du y Zhai, s.f.) utilizaron estados discretos y tres funciones de valor estado-acción diferentes. Jin y El-Saawy (Jiang y col., 2017) utilizaron entradas continuas pero describieron el espacio de acción para entrenar una red

neuronal que aproxima la función de valor estado-acción. El rendimiento de sus dos agentes varió significativamente dependiendo del tipo de función de valor Q (en el caso de Du et al. (Du y Zhai, s.f.)) o del tipo de métrica de rendimiento, la duración del historial de precios de las acciones y la penalización por volatilidad en la función de recompensa (en el caso de Jin y El-Saawy (Jiang y col., 2017)). heterogeneidad de rendimientos de las distintas carteras puede atribuirse al hecho de que la determinación de la política global óptima en una función de valor Q arbitraria es a menudo inviable sin una convexidad garantizada, y el Q-Learning también sufre de inestabilidad para la selección de la política óptima cuando ruido aleatorio en la función de pérdida. Esta aleatoriedad puede provenir de la dinámica estocástica o la observabilidad parcial del entorno, la incertidumbre aleatoria en la función de recompensa (incluyendo el error humano en el etiquetado), o la incertidumbre epistémica en la función de valor Q (por ejemplo, una mala aproximación de la función).

Los métodos de RL basados en políticas, en cambio, pueden aplicarse directamente a grandes dominios continuos. Sin embargo, la aproximación de la política óptima con una red neuronal con muchos parámetros (incluidos los hiperparámetros) es difícil y puede sufrir óptimos locales debido principalmente a su inestabilidad, ineficiencia de la muestra y sensibilidad a la selección de los valores de los hiperparámetros. Jiang et al. (Jiang y col., 2017) construyeron una red neuronal tipo DDPG con entradas de estado continuas para aproximar directamente la política óptima. Los pesos de la cartera producidos por su agente se asemejan a los de la cartera de bajo el criterio de Kelly, y las ponderaciones a menudo alternaban entre 0 y 1 en un corto periodo de tiempo, mostrando un alto grado de inestabilidad. Liang et al. (Liang y col., 2018) construyeron tres agentes de cartera diferentes utilizando cada uno el DPG, el DDPG y el hyperlinkPPOPPO con entradas de estado continuas, pero tanto los agentes DDPG como PPO no lograron aprender la política óptima en el entrenamiento a pesar de ser métodos más avanzados de Aprendizaje Profundo de Refuerzo (DRL) que el DPG.

Capítulo 3

Introducción al aprendizaje por refuerzo profundo

En este capítulo se explicará el marco teórico de trabajo del aprendizaje por refuerzo profundo. En primer lugar, se enunciará la base teórica que sustenta los Modelos de Decisión de Markov para, así, entender el punto de partida del RL. Una vez enunciada su piedra angular, se introducirán los elementos fundamentales de cualquier algoritmo de RL, así como, sus principales variantes y los modelos más comúnmente empleados en la práctica. El correcto entendimiento de este capítulo resulta esencial para poder aplicar dichos algoritmos al campo de las finanzas, así como, para comprender sus principales limitaciones.

3.1. Introducción a los elementos básicos del RL

Las técnicas de *Machine Learning* proporcionan métodos automatizados que pueden detectar patrones en los datos y utilizarlos para realizar numerosas tareas. Se distinguen tres tipos de tareas de aprendizaje automático:

- El *aprendizaje supervisado* que consiste en inferir una clasificación o regresión a partir de datos de entrenamiento previamente etiquetados.
- El *aprendizaje no supervisado* que realiza inferencias a partir de conjuntos de datos sin ningún tipo de etiqueta.
- El *aprendizaje por refuerzo* que aprende cómo los agentes toman una serie de acciones en un entorno para maximizar las recompensas acumuladas.

Para resolver estas tareas de aprendizaje automático se emplea la idea de funciones de aproximación. Existen diferentes tipos: modelos lineales, máquinas de soporte de vectores (SVM), árboles de decisión, procesos gaussianos, etc. En los últimos años, principalmente debido a los recientes desarrollos en el *Deep Learning* (DL) las técnicas de *Machine Learning* (ML) han experimentado mejoras espectaculares cuando se aprenden datos de alta dimensión como series temporales, imágenes y vídeos. Dichos avances se deben fundamentalmente a la congruencia de tres factores: el aumento exponencial de la potencia de cálculo con el uso de las GPUs y la computación distribuida, los avances metodológicos en el DL y, el creciente ecosistema de programas informáticos de código abierto como *TensorFlow* y conjunto de datos como *ImageNet*. Todos estos aspectos complementarios han dado lugar a un círculo virtuoso para el desarrollo del DL.

3.1.1. Marco teórico de trabajo

Un tema central en el DL es la toma de decisiones secuenciales. Esta es la tarea de decidir, a partir de la experiencia, la secuencia de acciones en un entorno incierto para alcanzar unos objetivos determinados. Dicha gestión abarca una amplia gama de posibles aplicaciones con el potencial de impactar en muchos dominios, como la robótica, la sanidad, las redes inteligentes, las finanzas y los coches autónomos, entre muchos de ellos. Inspirado en la psicología del comportamiento, el RL propone un marco formal para este problema. La idea principal es que un agente artificial puede aprender interactuando con su entorno, de forma similar a un agente biológico. Utilizando la experiencia acumulada, el agente debería de ser capaz de optimizar algunos objetivos dados en forma de recompensas acumulativas. En este punto, resulta importante matizar que el objetivo final es maximizar esa recompensa acumulada y, que por lo tanto, como las recompensas no son instantáneas, el agente debe razonar y elegir las acciones que a largo plazo le vayan a dar mejores resultados.

Este enfoque se aplica, en principio, a cualquier tipo de problema de toma de decisiones secuenciales que se base en la experiencia pasada. Se trata de un algoritmo dotado de una gran flexibilidad ya que el entorno puede ser estocástico, las observaciones pueden ser de alta dimensionalidad y el agente puede tener información imparcial sobre el entorno y puede recoger la experiencia del mismo basado en este o en cualquier otro tipo de datos.

El problema general del RL se formaliza como un proceso de control estocástico

en tiempo discreto en el que un agente interactúa con su entorno de la siguiente manera: el agente comienza en un estado dado dentro de su entorno $s_0 \in S$, recogiendo una observación inicial $\omega_0 \in \Omega$. En cada paso de tiempo t el agente tiene que realizar una acción $\in A$. Como se puede observar en la Figura 3.1 se tiene tres consecuencias:

- El agente obtiene una recompensa $r_t \in R$
- El estado transita a $s_{t+1} \in S$
- El agente obtiene una observación $\omega_{t+1} \in \Omega$

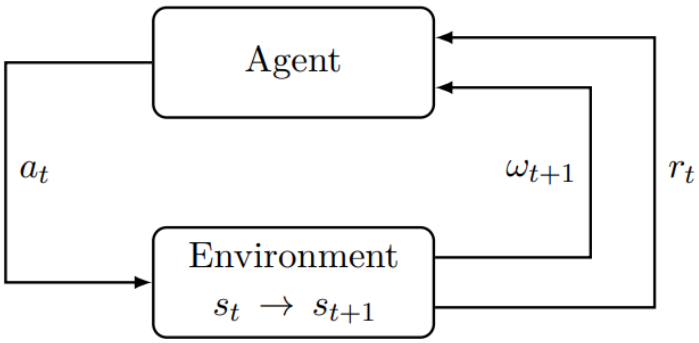


Figura 3.1. *Interacción agente-entorno en RL*

Esta configuración de control fue propuesta por primera vez por Bellman (Bellman, 1957) y posteriormente extendida al aprendizaje por Barto et al. (Barto y col., 1983). Un tratamiento exhaustivos de los fundamentos del RL en el son proporcionados por Sutton & Barto (Sutton y G, 2017).

3.1.2. Modelos de Decisión de Markov (MDP)

Para a empezar a abordar la cuestión se va a considerar el caso de los procesos de control estocásticos definidos por Norris (Norris, 1998). Un proceso estocástico en tiempo discreto posee la propiedad de Markov si:

- $\mathbb{P}(\omega_{t+1} \mid \omega_t, a_t) = \mathbb{P}(\omega_{t+1} \mid \omega_t, a_t, \ldots, , \omega_0, a_0)$
- $\mathbb{P}(r_t \mid \omega_t, a_t) = \mathbb{P}(r_t \mid \omega_t, a_t, \ldots, , \omega_0, a_0)$

La propiedad de Markov establece que el futuro del proceso sólo depende en la observación actual y que por lo tanto, el agente no tiene interés en mirar hacia atrás en toda la historia. Asimismo, un proceso de Markov de Decisión (MDP) (Bellman, 1957) es un proceso de control estocástico de tiempo discreto definido por una tupla de 5 elementos (S, A, T, R, γ) donde:

- S es el estado
- A es la acción en dicho espacio
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ es la función de transición, que se corresponde con las diferentes transiciones de cada estado.
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$ es la función de recompensa, donde R es un continua lista de posibles recompensas en un rango $R_{\text{máx}} \in \mathbb{R}^+$(e.g., $[0, R_{\text{máx}}]$)
- $\gamma \in [0, 1)$ es el factor de descuento que controla la importancia de los refuerzos a largo plazo: cuanto menor es el valor, menor peso se le otorga a las recompensas de las últimas etapas y viceversa.

El sistema es completamente observable en un MDP, lo que significa que la observación es igual al estado del entorno: $\omega_t = s_t$. En cualquier instante t la probabilidad de avanzar al estado s_{t+1} está representado por la función de transición $T(s_t, a_t, s_{t+1})$ y la recompensa viene dada por $R(s_t, a_t, s_{t+1}) \in \mathcal{R}$, tal y como se puede visualizar en la Figura 3.2.

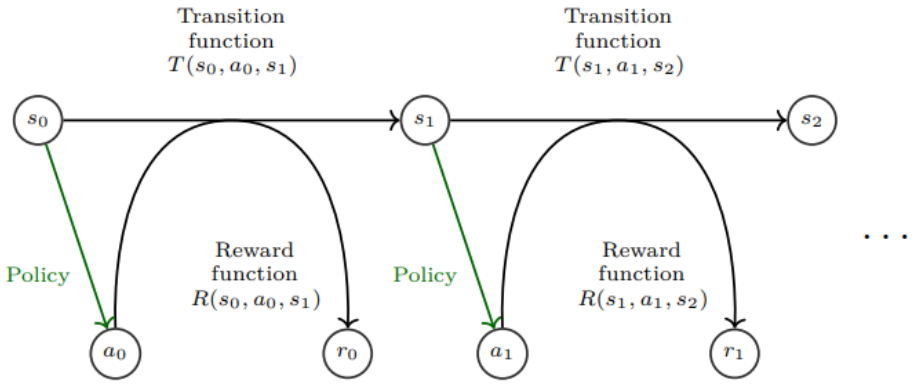


Figura 3.2. *Esquema del Proceso de Decisión de Markov*

3.1.3. Diferentes categorías de funciones de política

Una política define cómo un agente selecciona las acciones. Las políticas pueden ser estacionarias o no estacionarias. Una política no estacionaria depende del paso del tiempo y es útil para el contexto de horizonte finito en el que las recompensas acumuladas que el agente busca optimizar están limitadas a un número finito de pasos de tiempo. Un segundo criterio de clasificación consiste en discriminar las políticas por ser deterministas ($\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$) o estocásticas ($\pi(s,a) : \mathcal{S} \times \mathcal{A} \rightarrow$[0,1], donde $\pi(s,a)$ representa la probabilidad de que una acción a se sea elegida en un estado s.

3.1.4. Función de recompensa

El objetivo del agente consiste en encontrar una política $\pi(s,a) \in \Pi$ que optimice la recompensa esperada $V^{\pi}(s) : \mathcal{S} \rightarrow \mathbb{R}$, siendo $V^{\pi}(s)$:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right] \tag{3.1}$$

Donde:

- $r_t = \underset{a \sim \pi(s_t \cdot)}{\mathbb{E}} R\left(s_t, a, s_{t+1}\right)$
- $\mathbb{P}\left(s_{t+1} \mid s_t, a_t\right) = T\left(s_t, a_t, s_{t+1}\right)$ con $a_t \sim \pi\left(s_t, \cdot\right)$

En este sentido, la recompensa esperada se calcula para un número infinito de pasos, pero favoreciendo la recompensa a corto plazo. De este modo, el favorecimiento de la recompensa a corto plazo va a depender de un parámetro que se puede adaptar al problema en cuestión ya que habrá veces que interese favorecer la recompensa inmediata y otras veces que menos.Es por esto que este enfoque de la recompensa esperada recibe el nombre de recompensa descontada en horizonte infinito. De tal forma, la función de recompensa óptima se puede definir como:

$$V^{*}(s) = \underset{\pi \in \Pi}{\text{máx}}\, V^{\pi}(s) \tag{3.2}$$

Además de la función valor, hay otras funciones de especial interés como la función Q-valor, $Q^{\pi}(s,a) : \mathcal{S} \times A \rightarrow \mathbb{R}$, que se puede representar matemáticamente atendiendo a la siguiente ecuación:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right] \tag{3.3}$$

Esta función se puede definir de forma recursiva si se trata de un MDP utilizando la ecuación de Bellman:

$$Q^{\pi}(s,a) = \sum_{s' \in S} T\left(s,a,s'\right)\left(R\left(s,a,s'\right) + \gamma Q^{\pi}\left(s', a = \pi\left(s'\right)\right)\right) \tag{3.4}$$

De forma similar que para la función de recompensa óptima, existe una función Q-valor óptima definida por:

$$Q^{*}(s,a) = \max_{\pi \in \Pi} Q^{\pi}(s,a) \tag{3.5}$$

La diferencia entre $Q^{*}(s,a)$ y $V^{*}(s)$ es que la función política se puede obtener directamente de $Q^{*}(s,a)$:

$$\pi^{*}(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q^{*}(s,a) \tag{3.6}$$

La función de valor óptima $V^{*}(s)$ es la recompensa esperada descontada cuando se está en un estado s determinado mientras se sigue una política π^{*}. El valor óptimo de la función Q, $Q^{*}(s,a)$, es el la recompensa esperada descontada en un determinado estado y acción cuando se sigue la política π^{*}. De esta forma, se puede definir también la función de ventaja, que compara lo buena que es una acción a en comparación con la recompensa cuando se sigue directamente la política π, como:

$$A^{*}(s,a) = Q^{*}(s,a) - V^{*}(s) \tag{3.7}$$

Una forma directa de obtener las estimaciones de $V^{*}(s), Q^{*}(s,a), o, A^{*}(s,a)$ es utilizar los métodos de Monte Carlo, definiendo un estimador al establecer sucesivas simulaciones desde s siguiendo la función política π.

3.1.5. Diferentes componentes para aprender una política

La solución a un problema de RL viene dada como la asociación de una acción a cada estado de manera que se alcance el objetivo según el modelo de recompensa descontada en horizonte infinito. Esta asociación que define el comportamiento del agente es precisamente lo que se conoce como política. De forma global, un agente de RL incluye alguno de los siguientes componentes:

- Una representación de una *función valor* que proporciona una predicción de cómo de bueno es cada estado o cada par estado-acción.
- Una representación directa de la *política* $\pi(s)$ o $\pi(s,a)$.

- Un *modelo* del entorno junto con la estimación de la función de transición y de la de recompensa, en conjunción con los algoritmos necesarios.

Los dos primeros componentes están relacionados con lo que se denomina métodos sin modelo mientras que el último forma parte de la vertiente de métodos basados en modelos. La Figura 3.3 muestra un esquema de todas las variantes posibles.

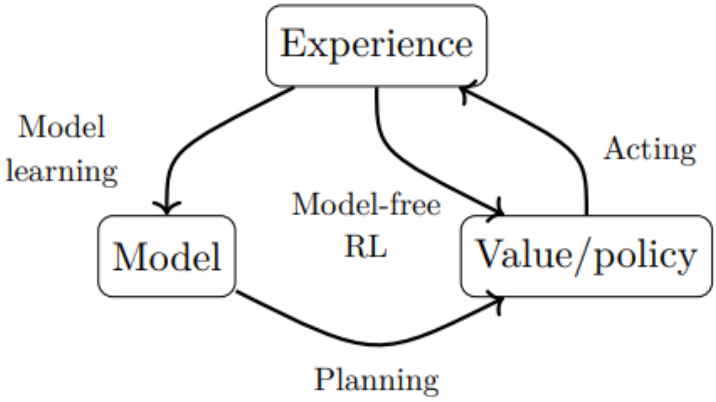


Figura 3.3. *Esquema general de los diferentes modelos de RL*

3.1.6. Diferentes configuraciones para aprender una política de los datos

1. Aprendizaje offline y online

El aprendizaje de una tarea secuencial de toma de decisiones aparece en dos casos: en el caso de aprendizaje offline en el que sólo se disponen de datos limitados sobre un entorno determinado y, en el caso de aprendizaje online en el que, paralelamente al aprendizaje, el agente va acumulando nueva experiencia en el entorno. En ambos casos, los algoritmos básicos de aprendizaje son los mismo. La particularidad de la configuración por lotes es que el agente tiene que aprender a partir de datos limitados sin la posibilidad de interactuar con el entorno. Sin embargo, en el aprendizaje online el principal problema es que el aprendizaje en sí es más complejo debido a la enorme cantidad de datos disponibles. De este modo, el agente tiene que optimizar la experiencia mediante el equilibrio entre la exploración y explotación. Para ello, puede utilizar una memoria de repetición que le

permite volver a procesar la información en un momento posterior. Un esquema general de los diferentes elementos del RL puede encontrarse en la Figura 3.4.

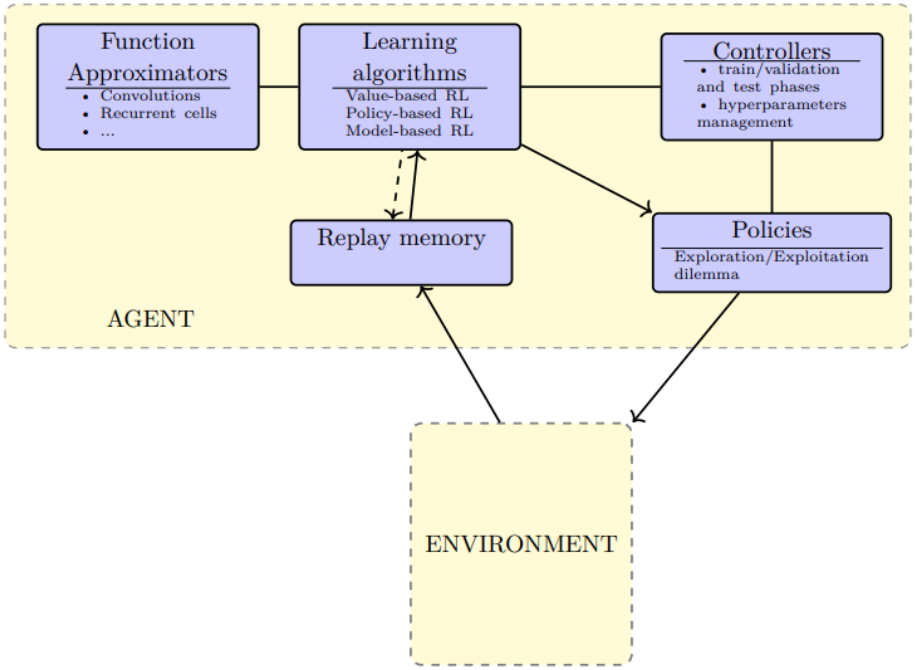


Figura 3.4. *Esquema general de los diferentes métodos de RL*

2. Aprendizaje off-policy y on-policy

Según Sutton & Barto (Sutton y G, 2017) los métodos *on-policy* intentan evaluar o mejorar la política que se utiliza para tomar decisiones, mientras que los métodos *off-policy* mejoran una política distinta de la que se utiliza para generar los datos. En los métodos basados en políticas (*on-policy*), el aprendizaje es directo cuando se utilizan trayectorias que no se obtienen necesariamente bajo la política actual, sino a partir de una política de comportamiento diferente $\beta(s, a)$. En estos casos, la repetición de la experiencia permite reutilizar muestras de una política de comportamiento diferente. Sin embargo, los métodos basados en políticas suelen introducir un sesgo cuando se utilizan con un búfer de repetición, ya que las trayectorias no suelen obtenerse únicamente bajo la política actual π. Este fenómeno hace que los métodos *off-policy* sean eficientes, ya que son capaces de aprovechar cualquier tipo de experiencia sin introducir ningún tipo de sesgo adicional.

3.2. Métodos basados en valores para RL

Los algoritmos basados en valores tienen como objetivo construir una función de valor que posteriormente permite definir una política. En lo sucesivo, se discutirán las bases teóricas que sustentan el desarrollo de cada uno de estos algoritmos tomando por referencia el más sencillo de los mismos, Q-Learning. Por el teorema de *Banah*, el punto fijo del operador de Bellman existe ya que es un mapeo de contracción. [1]. Este algoritmo no suele ser aplicable de forma práctica debido al gran número de pares estado-acción disponibles. En este sentido, se necesita una versión parametrizada de la función Q, $Q(s, a,)$

3.2.1. Q-Learning

La versión más simple de este algoritmo tiene una tabla de valores $Q(s, a)$ para cada par cuya única solución es $Q^*(s, a)$:

$$Q^*(s, a) = (\mathcal{B}Q^*)(s, a) \tag{3.8}$$

Donde $\mathcal{B}$ es el operador de Bellman que mapea cualquier función $K : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ con otra función $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ definida mediante la siguiente ecuación:

$$(\mathcal{B}K)(s, a) = \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} K(s', a') \right) \tag{3.9}$$

Esta configuración simple es a menudo inaplicable debido al espacio de estado-acción de alta dimensión (posiblemente continuo). En ese contexto, se necesita una función de valor parametrizada $Q(s, a; \theta)$, donde θ se refiere a algunos parámetros que definen los valores Q.

3.2.2. Q-Learning ajustado

En este tipo de aprendizaje ajustado, el algoritmo empieza con una inicialización aleatoria de valores $Q(s, a; \theta_0)$ donde θ_0 hace referencia a los parámetros iniciales. A continuación, se actualizan los Q-valores en cada iteración k según la siguiente relación:

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k) \tag{3.10}$$

[1] El operador de Bellman es un mapeo de contracción porque se puede demostrar que para cualquier par de funciones acotadas $K, K' : S \times A \to \mathbb{R}$ se cumple la siguiente condición: $\|TK - TK'\|_\infty \leq \gamma \|K - K'\|_\infty$

En el NFQ, *neural fitted Q-Learning*, el estado puede ser proporcionado como entrada a la red Q y se da una salida diferente para cada acción posible. Esto proporciona una estructura eficiente que tiene la ventaja de obtener el máximo de $Q(s, a; \theta_0)$ mediante una sola computación del *forward-pass* de la red neuronal. En este sentido, los Q-valores mencionados anteriormente se parametrizan con una red neuronal $Q(s, a; \theta_k)$ donde θ_k se actualiza aplicando la técnica del descenso por gradiente, minimizando la siguiente función de coste:

$$\mathrm{L}_{DQN} = \left(Q(s, a; \theta_k) - Y_k^Q\right)^2 \tag{3.11}$$

La principal desventaja de este método es que al actualizar los pesos también se modifica la función objetivo y por ello, debido a la capacidad de generalización y extrapolación de las redes neuronales, este enfoque puede generar grandes errores en diferentes combinaciones del espacio estado-acción, convirtiéndose entonces en un modelo con cierta inestabilidad.

3.2.3. Q-Learning profundo (DQN)

Aprovechando las ideas de NFQ, el algoritmo de la red Q profunda (DQN), introducido por Mnih et al. (Mnih y col., 2015), es capaz de obtener un gran rendimiento en entornos online usados para multitud de juegos ATARI. Utiliza tres técnicas heurísticas para combatir las inestabilidades de los métodos anteriores:

1. Se reemplaza la red neuronal Q por $Q\left(s', a'; \theta_k^-\right)$ donde los parámetros se actualizan únicamente cada $C \in \mathbb{N}$ iteraciones. Esto previene que las inestabilidades se propaguen de forma rápida y reduce el riesgo de la divergencia de la función objetivo. La idea de las redes objetivo puede entenderse como una instancia del Q-learning, donde cada período entre las actualizaciones de la red objetivo corresponde a una única iteración Q ajustada.

2. En un aprendizaje online la memoria mantiene toda la información para las $N_{replay} \in \mathbb{N}$ pasos donde la experiencia se guarda con una política $\epsilon greedy$. De este modo, las actualizaciones se hacen de forma aleatoria mediante mini-batches permitiendo que se cubran un mayor rango de tuplas estado-acción en el espacio de soluciones.

3. Además de la modificación de la red y la memoria de repetición, DQN escala los valores de la función objetivo para garantizar un aprendizaje adecuado en la práctica de las recompensas, esto limita las derivadas del error y facilita

el uso de la misma tasa de aprendizaje en distintos escenarios.

En la Figura 3.5 se adjunta un esquema del método descrito.

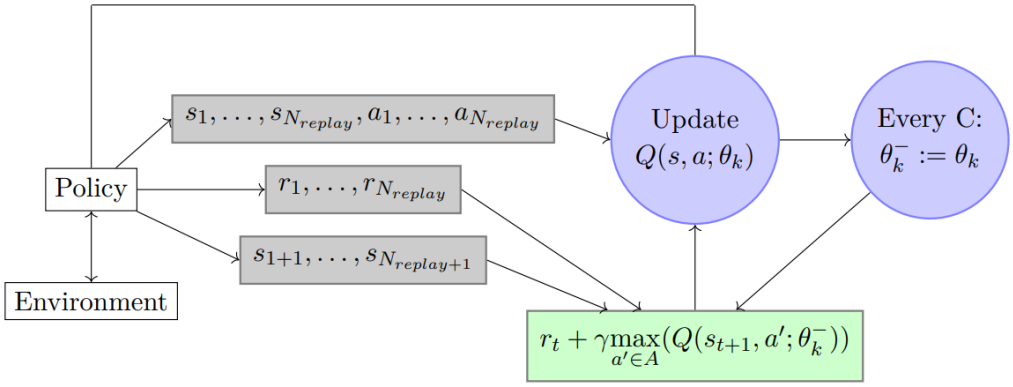


Figura 3.5. *Esquema del algoritmo DQN*

3.2.4. Doble DQN

Como se ha explicado anteriormente, la operación para encontrar el máximo de $Q(s, a; \theta)$ emplea los mismos valores tanto para seleccionar como para evaluar una acción. Esto hace que sea más probable seleccionar valores sobreestimados en caso de imprecisiones o ruido, lo que genera estimaciones de valores muy optimistas. Por lo tanto, el algoritmo DQN induce un sesgo al alza. De este modo, el método del doble estimador emplea dos estimadores diferentes para cada variable lo que permite desacoplar la selección de un estimador y su valor, gracias a ello, independientemente de si los errores en los valores Q estimados se deban a la estocasticidad del entorno en la aproximación de la función, a la no estacionariedad o a cualquier otra fuente plausible, este método permite eliminar el sesgo positivo en la estimación de los valores de cada acción. Matemáticamente, en la DDQN el valor objetivo Y_k^Des reemplazado por:

$$Y_k^{DDQN} = r + \gamma Q\left(s', \underset{a \in \mathcal{A}}{\text{argmax}} Q\left(s', a; \theta_k\right); \theta_k^-\right) \tag{3.12}$$

Lo que permite una reducción de la sobreestimación de los Q valores y mejora por tanto, la estabilidad del método.

3.2.5. DQN distribucional

Todos los enfoques descritos hasta ahora en el presente capítulo aproximan directamente la rentabilidad esperada de una función valor. Otra alternativa consiste en buscar una representación más rica a través de una distribución de valores, es decir mediante la distribución de los posibles rendimientos acumulados. Esta proporciona una información más completa de la aleatoriedad intrínseca de las recompensas y las transiciones del agente dentro de su entorno. La distribución de valores Z_π es un mapeo de pares estado-acción a distribuciones de rendimientos cuando se sigue la política π. Este rendimiento se describe, como sigue, mediante una ecuación recursiva de naturaleza distribucional:

$$Z^\pi(s,a) = R\left(s,a,S'\right) + \gamma Z^\pi\left(S',A'\right) \tag{3.13}$$

La ecuación de Bellman distribucional establece que la distribución Z se caracteriza mediante la interacción de tres variables aleatorias: la función recompensa, $R\left(s,a;S'\right)$, el siguiente par estado-acción $\left(S',A\right)$ y el rendimiento $Z^\pi\left(S',A'\right)$. En numerosos estudios como el propuesto por Bellemare et al (DBQN) se prueba que este enfoque distribucional puede ser empleado en la práctica como función para aproximar los Q-valores trayendo consigo dos ventajas principales:

1. La posibilidad de aplicar un comportamiento consciente del riesgo de cada acción.

2. La capacidad de dotar de un aprendizaje más eficaz ya que ofrece un conjunto más rico de señales de entrenamiento que una simple función $Q(s,a)$. Aunque estas señales no son necesarias a priori para optimizar el rendimiento esperado, se conocen como tareas auxiliares y conducen a un aprendizaje mejorado.

En la Figura 3.6 se adjunta una comparación enfoque MDP (izquierda) con la estimación de Q-valores distribucionales (derecha).

3.3. Métodos basados en políticas para RL

Esta sección se centra en una familia particular de algoritmos de aprendizaje por refuerzo que utilizan métodos de gradiente de políticas. Estos métodos optimizan una función objetivo (normalmente la recompensa acumulada esperada) encontrando una buena política gracias a variantes del gradiente estocástico con respecto a los parámetros de la política.

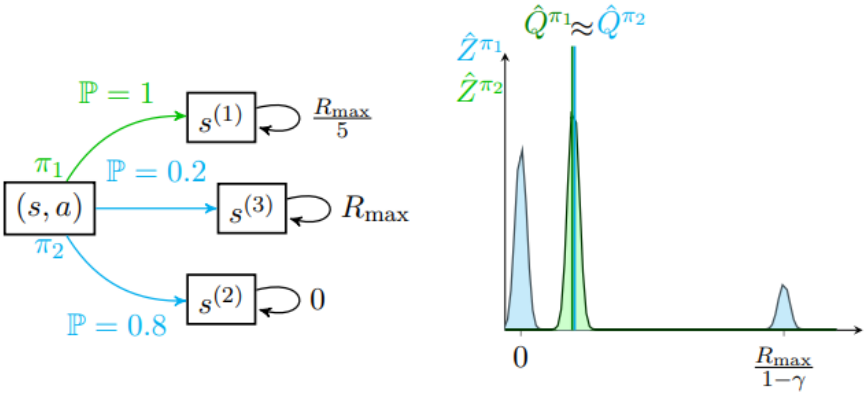


Figura 3.6. *Comparación del enfoque distribucional*

3.3.1. Políticas estocásticas

La recompensa esperada de una política estocástica π empezando en un estado determinado s_0 puede representarse matemáticamente mediante la siguiente ecuación:

$$V^{\pi}(s_0) = \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi(s,a) R'(s,a) da ds \tag{3.14}$$

donde

$$R'(s,a) = \int_{s' \in \mathcal{S}} T(s,a,s') R(s,a,s')$$

y $\rho^{\pi}(s)$ son la distribución de estado descontada definida por:

$$\rho^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \Pr\{s_t = s \mid s_0, \pi\} \tag{3.15}$$

Para una política diferenciable π_w el fundamento teórico que subyace bajo estos algoritmos es el teorema de la política del gradiente, definido como sigue:

$$\nabla_w V^{\pi_w}(s_0) = \int_{\mathcal{S}} \rho^{\pi_w}(s) \int_{\mathcal{A}} \nabla_w \pi_w(s,a) Q^{\pi_w}(s,a) da ds \tag{3.16}$$

Lo que permite adaptar los parámetros de la política según la experiencia. Este resultado es particularmente interesante ya que la política del gradiente no depende del gradiente de la distribución del estado. De este modo, la forma más sencilla de derivar la política del estimador del gradiente es mediante el paradigma de la razón de verosimilitud, a través de la siguiente fórmula:

$$\begin{aligned}\nabla_w \pi_w(s,a) &= \pi_w(s,a)\frac{\nabla_w \pi_w(s,a)}{\pi_w(s,a)} \\ &= \pi_w(s,a)\nabla_w \log\left(\pi_w(s,a)\right)\end{aligned} \tag{3.17}$$

Hasta ahora, se ha demostrado que los métodos de gradiente de políticas deben incluir una evaluación de la política seguida de una mejora de la misma. Por un lado, la evaluación de la política estima Q^{π_w} y por otro lado, la mejora de la política da un paso en favor del gradiente para optimizar la política $\pi_w(s,a)$ con respecto a la estimación del valor objetivo. De forma intuitiva, la mejora de la política en cada paso aumenta la probabilidad de las acciones proporcionalmente a su recompensa esperada. Sin embargo, la cuestión que queda por abordar es cómo el agente puede realizar el paso de evaluación de la política, es decir, cómo se obtiene una estimación Q^{π_w}. El enfoque más sencillo para estimar los gradientes es sustituir el estimador de la función Q por un estimador que acumule la recompensa de las trayectorias completas. En el gradiente de política de Monte Carlo, se estima mediante los rodajes en el entorno mientras se sigue la política π_w. Se trata de una estimación sin sesgo cuando se emplea junto con la retropropagación de una política de red neuronal ya que estima los retornos hasta el final de las trayectorias, evitando así las inestabilidades inducidas por el bootstrapping. Sin embargo, el principal inconveniente es que la estimación requiere el despliegue en la política y puede presentar una alta varianza. Por lo general, se necesitan varias pruebas para obtener una buena estimación, por lo tanto, es más común emplear una estimación de la recompensa con un enfoque basado en valores como los métodos actor-crítico.

Para finalizar, es necesario mencionar dos últimos matices. En primer lugar, para evitar que la política sea determinista, es habitual añadir un regularizador de entropía al gradiente, lo que garantiza que la política siga explorando. En segundo lugar, en vez de la función de valor, Q^{π_w} también se puede emplear una función de ventaja A^{π_w}. La diferencia principal entre ambas es que, mientras que $Q^{\pi_w}(s,a)$ resume el rendimiento de cada acción para un estado determinado bajo la política π_w, la función de ventaja $A^{\pi_w}(s,a)$ proporciona una medida de comparación para cada acción con el rendimiento esperado en el estado s, dado por V^{π_w}. El uso de $A^{\pi_w}(s,a)$ hace que las magnitudes sean más bajas y esto ayuda a reducir la varianza del estimador del gradiente.

3.3.2. Políticas deterministas

Los métodos de gradiente de políticas pueden extenderse a las políticas deterministas. La NFQ de acciones continuas (NFQCA) y el DDPG introducen la representación directa de una política de tal manera que se pueden ampliar los algoritmos NFQ y DQN para superar la restricción de acciones discretas. Si se denota por $\pi(s)$ la política determinista $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$. En espacios discretos un enfoque directo es construir una política de forma iterativa mediante:

$$\pi_{k+1}(s) = \underset{a \in \mathcal{A}}{\text{argmax}} Q^{\pi_k}(s, a) \tag{3.18}$$

donde π_k es la política en la k iteración. Sin embargo, en un espacio continuo, una política *greedy* es un tanto problemática ya que requiere una maximización global en cada paso. De este modo, se emplea en su lugar $\pi_k(s)$ que es una política determinista diferenciable que da lugar al DDPG, representado matemáticamente mediante la siguiente ecuación:

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_w}} \left[\nabla_w (\pi_w) \nabla_a (Q^{\pi_w}(s, a))|_{a=\pi_w(s)} \right] \tag{3.19}$$

Esta ecuación lleva implícito la confianza en $\nabla_a (Q^{\pi_w}(s, a))$ que suele requerir el uso de métodos de actor-crítico que se explican a continuación.

3.3.3. Métodos actor-crítico

Como se ha visto en las secciones anteriores, una política representada por una red neuronal puede ser actualizada mediante el ascenso por el gradiente tanto en políticas deterministas como estocásticas. En ambos casos, la política del gradiente típicamente requiere un estimador del valor de la función para la política actual. Un enfoque común consiste en emplear la arquitectura del actor-crítico que se basa fundamentalmente en dos partes: un actor, que hace referencia a la política y un crítico, que representa el valor de una función, por ejemplo, Q-valor. En el aprendizaje por refuerzo profundo tanto el actor como el crítico se representan mediante funciones de aproximación no lineales con el uso de redes neuronales. El crítico, parametrizado mediante θ, estima el valor aproximado de la función de la política actual $\pi : Q(s, a; \theta) \approx Q^{\pi}(s, a)$. A continuación, se explican en detalle cada uno de los agentes anteriores.

a) El crítico

A partir de una tupla extraída de la memoria de repetición, el enfoque *off-policy* más simple consiste en estimar el crítico utilizando un algoritmo de *bootstraping* puro ($TD(0)$) donde en cada iteración, el valor actual $Q(s, a; \theta)$ se actualiza según la siguiente expresión:

$$Y_k^Q = r + \gamma Q\left(s', a = \pi\left(s'\right); \theta\right) \tag{3.20}$$

Este paradigma tiene como principal ventaja su simplicidad, sin embargo, como carencia su coste computacional, al igual que sucedía con los métodos basados en valores discutidos anteriormente. De forma general, la arquitectura ideal es aquella que:

- Es eficiente en cuanto al muestreo, de manera que pueda utilizar tanto trayectorias fuera de la política como dentro de la misma, es decir, que sea capaz de adoptar una memoria de repetición.
- Es computacionalmente eficiente, es decir, debe ser capaz de beneficiarse de la estabilidad y la rápida propagación de la recompensa de los métodos *on-policy* para las muestras recogidas de las políticas de comportamientos cercanas.

b) El actor

Como ya se ha discutido anteriormente, el *off-policy gradient* en la mejora de la política para el caso estocástico puede definirse como:

$$\nabla_w V^{\pi_w}\left(s_0\right) = \mathbb{E}_{s\sim\rho^{\pi_\beta}, a\sim\pi_\beta}\left[\nabla_\theta\left(\log \pi_w(s, a)\right) Q^{\pi_w}(s, a)\right] \tag{3.21}$$

donde β es una política de comportamiento generalmente diferente a π lo que sesga enormemente al gradiente. Este enfoque suele comportarse correctamente en la práctica, sin embargo, el uso de un estimador de gradiente de política sesgado dificulta el análisis de su convergencia.

En el caso de los métodos actor-crítico se ha investigado un enfoque para realizar la política gradiente sin repetición de la experiencia con el uso de métodos asíncronos en los que se ejecutan múltiples agentes en paralelo y los actores se entrenan de forma aislada. La paralelización de los agentes también garantiza que cada uno de ellos experimente diferentes partes del entorno en un paso de tiempo determinado. En ese caso, se pueden utilizar retornos de n pasos sin introducir un sesgo. Esta sencilla idea puede aplicarse a cualquier algoritmo de aprendizaje que

requiera datos sobre la política y elimina la necesidad de mantener un búfer de repetición.

Por último, una alternativa es combinar las muestras fuera de la política y dentro de la misma para incrementar la eficiencia de la muestra de ambos métodos y la estabilidad de las estimaciones del gradiente sobre la política. Por ejemplo, *Q-Prop* utiliza un estimador de gradiente *on-policy* de Monte Carlo al tiempo que reduce la varianza del mismo mediante el uso de un crítico fuera de la política como variante de control. Existen otros métodos como políticas de gradientes naturales y la optimización de la región de confianza que permiten optimizar los métodos de la política de forma controlada.

3.4. Métodos basados en modelos para RL

En las dos secciones anteriores se han discutido diferentes enfoques de métodos de RL sin modelo que se basan bien en métodos de valores o en métodos de política. A continuación, se introducen los principales métodos de RL basados en un modelo del entorno (dinámica y función de recompensa) junto con un algoritmo de planificación.

3.4.1. Búsqueda lookahead

Este tipo de algoritmos construye iterativamente un árbol de decisión donde el estado actual es el nodo raíz que almacena los resultados obtenidos en el resto de los nodos y centra la atención en las trayectorias potenciales prometedoras. La principal dificultad en el muestreo de trayectorias es equilibrar la exploración y la explotación. Por un lado, el propósito de la exploración es reunir información en la parte del árbol de búsqueda en la que se han realizado pocas simulaciones, y por tanto, el valor esperado tiene alta varianza. Por el otro lado, el propósito de la explotación consiste en refinar el valor esperado de los movimientos más prometedores. La idea es muestrear múltiples trayectorias desde el estado actual hasta alcanzar una condición terminal, tal y como se puede apreciar en la Figura 3.7.

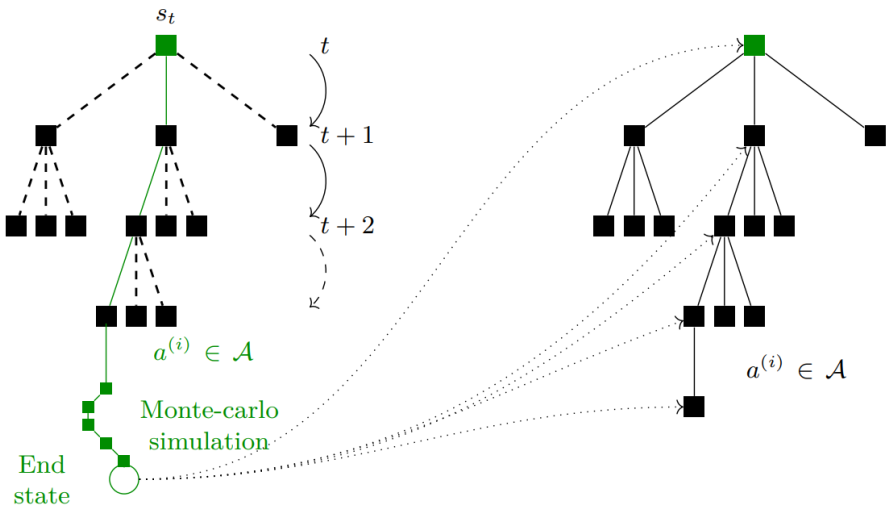


Figura 3.7. *Construcción de un árbol de decisión mediante un MCTS*

3.4.2. Optimización de la trayectoria

Las técnicas anteriores están limitadas a acciones discretas, su alternativa para espacios continuos, si el modelo es diferenciable, consiste en calcular directamente una política mediante la retropropagación de las recompensas a lo largo de las trayectorias. Por ejemplo, PILCO utiliza procesos gaussianos para aprender un modelo probabilístico de la dinámica. A continuación, puede utilizar explícitamente la incertidumbre para la planificación y evaluación de políticas para lograr una buena eficiencia de la muestra. Sin embargo, los procesos gaussianos no han sido capaces de escalar de forma fiable a problemas de alta dimensión. Un enfoque para escalar la planificación a dimensiones más altas es apuntar a aprovechar las capacidades de generalización del aprendizaje profundo.

Una vez familiarizados con el funcionamiento básico y principales posibilidades de los modelos de RL, en el siguiente capítulo se tratará el caso especial del RL aplicado al ámbito de la gestión de carteras.

Capítulo 4

Adaptación de las técnicas de RL a la gestión de una cartera

El objetivo fundamental de este capítulo es presentar la metodología desarrollada para la adaptación de las técnicas de aprendizaje por refuerzo profundo descritas en el capítulo anterior al problema que se pretende resolver. Para ello, se dividirá el contenido del mismo en seis partes, que se corresponden con cada uno de los pasos que se desarrollan de forma secuencial para construir la estrategia de inversión deseada. En primer lugar, se enunciará el problema en cuestión y sus principales limitaciones. A continuación, se hará una breve revisión de las librerías de *Python* empleadas, así como, de sus principales prestaciones. Posteriormente, se discutirá el método de extracción y preparación de los datos para entrenar el modelo. Finalmente, con los datos ya recopilados, se implementará el entorno virtual para la gestión automática de la cartera de inversión, junto con los diferentes algoritmos de RL explicados anteriormente.

4.1. Definición del problema

Como se ha explicado anteriormente, el problema de la asignación de capital de un portfolio determinado se puede describir como un Proceso de Decisión de Markov (MDP). Por esta razón, resulta fundamental entender qué significado adquiere la tupla de 5 elementos (S, A, T, R, γ) ya introducida en capítulos anteriores. De este modo, bajo el paradigma del entorno financiero cada una de las variables anteriores representan lo siguiente:

- Estado: esta variable viene definida por dos subconjuntos de elementos. En

primer lugar, por la matriz de covarianza de los retornos esperados para cada una de las acciones computados con una ventana móvil de 1 año. En segundo lugar, por un conjunto de indicadores técnicos que pretenden resumir la información completa del mercado. Entre ellos se encuentran:

- RSI: se trata de una herramienta clave del análisis técnico, que permite evaluar la tendencia de los activos para determinar si se encuentra en una zona de sobrecompra o de sobreventa.
- MACD: consiste en una media móvil de convergencia/divergencia que trata de identificar cambios en el impulso del precio de una acción.
- CCI: se trata de un indicador versátil que puede ser empleado para identificar una nueva tendencia en el mercado o advertir acerca de condiciones extremas.
- ADX: consiste en un indicador diseñado para medir la fuerza y dirección de la tendencia del mercado.

Posteriormente se discutirá el uso de un autoencoder LSTM que permita incluir un mayor número de indicadores técnicos y que mediante la reducción de dimensionalidad perimitiendo la extracción de las características más importantes de los mismos. Por último, es importante notar, que tanto la matriz de covarianza como el conjunto de indicadores representan el estado del día actual únicamente, lo que tiene sentido si se asume la propiedad Markoviana en el proceso. De este modo, se deja como línea de investigación para futuros desarrollos la adopción de estados que recojan una ventana móviles de datos para el agente.

- Acción: en este entorno las acciones que el agente debe tomar para maximizar la recompensa son precisamente las ponderaciones que se le otorgan a cada componente del índice en la cartera. Es importante notar que cada peso está restringido a tomar valores entre $[0, 1]$ ya que no se permite las posiciones en corto. Además, es preciso efectuar una normalización posterior para que la suma de todas las acciones resulten en la unidad, es decir, para que se invierta el 100 % de la cartera. Para ello, se emplea una normalización exponencial aplicando la función *softmax*:

$$\begin{aligned} &\sigma : \mathbb{R}^K \to [0,1]^K \\ &\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{para } j = 1, \ldots, K \end{aligned} \tag{4.1}$$

siendo K el número de constituyentes del índice Dow Jones en el periodo comprendido.

- Función de recompensa: en este caso se incluyen:

  1. El valor acumulado del portfolio como métrica de la rentabilidad final esperada tras aplicar la política. Para computarlo se tienen en cuenta unos costes de transacción del 0.5 % por cada operación efectuada.
  2. El ratio de Sharpe anualizado para tener en cuenta el riesgo de las acciones que el agente toma en cada paso.

  Se deja a elección del usuario la función con la que desea optimizar la actuación del agente.

- Entorno: se define un entorno personalizado heredando la clase *gym* disponible en la librería *OpenAI*. Se trata de un simulador que permite entrenar los agentes de DRL. La estructura básica de un entorno está compuesta por:

  1. Un espacio de observaciones que define la dimensión de las mismas, así como los umbrales de los valores de cada variable que conforma las observaciones del entorno. En el caso de estudio, está representado por una matriz de (34×30), donde el número 30 corresponde a las acciones que constituyen el índice (Dow Jones 30) y el 34 procede de los 4 indicadores técnicos descritos anteriormente y los treinta restante de los elementos de la matriz de covarianza para cada par de acción. En principio, los umbrales de las variables de entrada no se encuentran acotados.
  2. Un espacio de acciones definidos por los pesos ya descritos anteriormente

  Para la interacción con el entorno la clase tiene una serie de métodos que permite al agente completar su función:

  1. *Reset*: esta función permite resetear el entorno a su estado inicial y devuelve la observación del entorno correspondiente con dicho estado. En este punto conviene introducir los dos métodos de inicialización implementados. Se permite la inicialización de los pesos de dos maneras: (i) simplemente un portfolio equiponderado, donde el peso de cada acción en la cartera será $1/N = 1/30$; y (ii) mediante la solución de un problema de Media-Varianza (Markowitz). En secciones posteriores se analizará el impacto de la decisión de adoptar un punto de partida u otro.

2. *Step*: este método toma como entrada una acción y la aplica al entorno y es lo que permite evolucionar hacia el siguiente estado. En el caso de estudio, la función recibe como input un vector de ponderaciones determinado en un día t, y permite que el entorno transite al siguiente día de cotización. Aquí es importante advertir que para realizar una implementación más realista del método se ha dejado como hiperparámetro adicional el factor de rebalanceo de la cartera. Como consecuencia de esto, la unidad de transición de un estado a otro no siempre se corresponden con días laborales consecutivos.
3. *Observation*: se corresponde con la observación ya descrita en el punto anterior.
4. *Reward*: se corresponde con el beneficio que el agente adquiere del entorno al aplicar la acción que la función *step* recibe por entrada.
5. *Done*: se trata de una variable binaria que establece si el episodio actual ha finalizado. En el método descrito se configura con la última fecha disponible en los datos liberados para testear la estrategia de trading.
6. *Info*: proporciona información adicional como el número de iteraciones restantes o el camino hasta la convergencia del método.

4.2. Introducción a FinRL

FinRL, *Financial Reinforcement Learning Framework* [1], es el primer marco de trabajo *Open Source* que permite desplegar el potencial de aplicar aprendizaje por refuerzo profundo en el área de las finanzas cuantitativas. FinRL proporciona tres capas fundamentales: entornos, agentes y aplicaciones, como se puede visualizar en la Figura 4.1. De todas las funcionalidades disponibles, en el presente proyecto se ha hecho uso de los siguientes módulos:

- *Benchmark Test*: para simular el comportamiento de estrategias de inversión en horizontes temporales no observados por el modelo y, así poder analizar la bondad del método implementado.
- *Portfolio Allocation*: para diseñar el entorno de RL. Se ha tomado como semilla inicial y se han incluido las siguientes funcionalidades para hacerlo más realista:

[1] https://github.com/AI4Finance-Foundation/FinRL

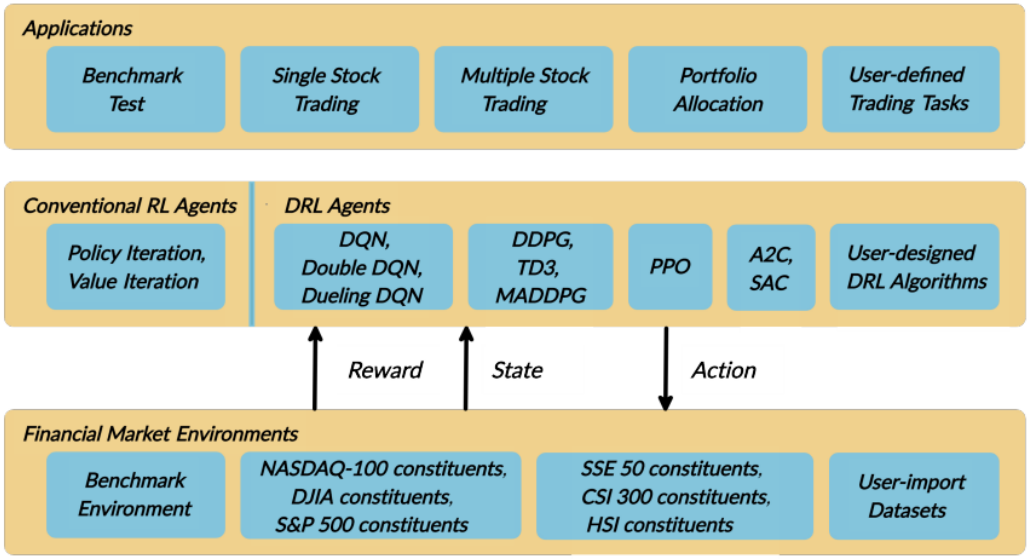


Figura 4.1. *Capas disponibles en FinRL*

1. Incluir los costes de transacción en el cálculo de la rentabilidad del modelo.
2. Permitir definir funciones de recompensa personalizadas como el ratio de Sharpe anualizado.
3. Habilitar la inicialización de los pesos del primer día mediante Markowitz.
4. Implementar un periodo de rebalanceo distinto a la unidad.
5. Dotar al agente de mayor información de cada estado mediante el empleo de un autoencoder LSTM que permite trabajar con un conjunto más amplio de indicadores técnicos.

- *DRL Agents: A2C, PPO, SAC, DDPG, TD3*: se han empleado para entrenar los modelos de RL. En las siguientes secciones se explicará de forma más detallada las particularidades de cada método. Es importante matizar que, de entre los algoritmos presentes, se han elegido únicamente los que permitían que el par de estado-acción fuese definido por variables continuas y no únicamente discretas.
- *DJIA constituents*: con el objetivo de acceder a la lista de acciones que constituyen el índice para la construcción de la base de datos.

- *Benchmark environment*: con el fin de establecer como uno de los benchmark la estrategia de Buy & Hold de un ETF del índice.

4.3. Descarga y extracción de los datos

Para la construcción inicial de la base de datos, se extrae de *Yahoo Finance* las cotizaciones de las acciones del índice a lo largo del periodo comprendido entre el 1 de Enero de 2008 y el 28 de Febrero de 2022. Es importante advertir que la lista de dichos valores permanece fija en el tiempo, es decir, que si se desea realizar un algoritmo más sofisticado se debería de tener en cuenta el sesgo de supervivencia e incluir acciones que, por las revisiones periódicas del índice, ya no formen parte del mismo. De esta forma, se obtiene un *dataset* que tiene por columnas las siguientes variables: *date*, *open*, *high*, *low*, *close*, *adjcp*, *volume*, *tic*, guardando el registro en formato *long*, tal y como se puede observar en la Figura 4.2. En la Figura 4.3 se muestra la evolución del precio de AAPL en el último trimestre, con el objetivo de garantizar la correcta lectura de la evolución histórica de los activos.

	date	open	high	low	close	adjcp	volume	tic	day
0	2006-01-03	2.585000	2.669643	2.580357	2.669643	2.285946	807234400	AAPL	1
1	2006-01-03	79.360001	81.239998	78.099998	80.360001	61.612621	9114400	AMGN	1
2	2006-01-03	51.700001	52.580002	51.049999	52.580002	40.728054	7825700	AXP	1
3	2006-01-03	70.400002	70.599998	69.330002	70.440002	50.119705	4943000	BA	1
4	2006-01-03	57.869999	58.110001	57.049999	57.799999	37.493549	3697500	CAT	1

Figura 4.2. *Dataset inicial*

4.4. Preprocesado de los datos

Los datos son el auténtico combustible con el que se alimenta la IA, por lo tanto, para que un modelo funcione de forma correcta es necesario un pretratamiento de los mismos. Según la metodología CRISP-DM (*Cross Industry Standard Process for Data Mining*) el preprocesado de los datos comienza con la limpieza de los mismos. Existen diferentes métodos para imputar valores erróneos presentes en los datos, sin embargo, en el marco de este proyecto no ha sido necesario utilizar

Figura 4.3. *Evolución del precio de AAPL en el último trimestre*

ninguno de los mismos ya que las series históricas de los precios estaban calculadas de forma correcta. No osbtante, los datos generados hasta ahora no están homogeneizados y por lo tanto, no son comparables entre sí.Esta es una de las razones por la que, en lugar de trabajar con las cotizaciones en crudo de los activos, se opera con la evolución de los indicadores técnicos de los mismos, que sí que son comparables entre sí. De este modo, el dataset inicial se amplía con las variables adjuntas en la Figura 4.4.

Asimismo, se enriquece los datos añadiendo la matriz de covarianza de los retornos anuales de cada acción, al ser una métrica muy empleada por los profesionales del sector. La Figura 4.5 representa la matriz de correlaciones que guarda una estrecha relación con la anterior y permite visualizar si existe o no una clara

	date	tic	day	macd	boll_ub	boll_lb	rsi_30	cci_30	dx_30	close_30_sma	close_60_sma
0	2007-01-04	AAPL	3	-0.027530	3.280724	2.849455	53.720136	-51.275262	27.932017	3.119893	2.984839
1	2007-01-04	AMGN	3	-0.639541	71.043735	67.872265	51.737860	27.733315	20.689603	70.195667	72.124333
2	2007-01-04	AXP	3	0.396697	62.422316	58.741683	54.209365	-3.440645	20.731259	60.135667	59.075500
3	2007-01-04	BA	3	0.425465	91.068842	88.407156	57.422488	-76.326080	4.320156	89.557000	86.244833
4	2007-01-04	CAT	3	-0.277834	63.434413	60.141587	44.818973	-102.843554	3.377121	61.828667	62.475833

Figura 4.4. *Ejemplo dataset con los indicadores técnicos*

multicolinearidad entre los datos disponibles.

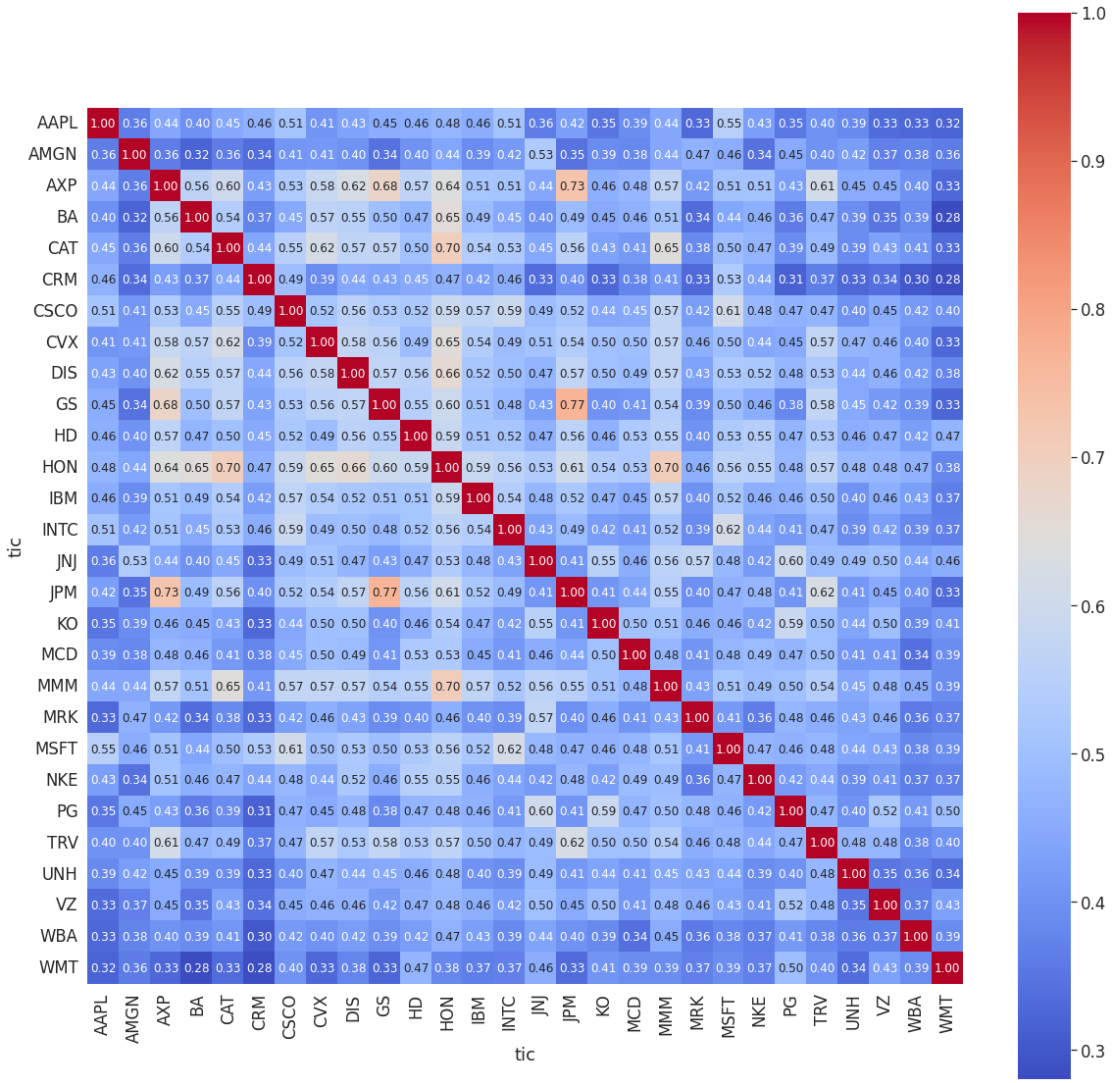


Figura 4.5. *Ejemplo matriz de correlaciones de cada uno de los activos*

4.4.1. Diseño de un autoencoder LSTM

Parece intuitivo pensar que a mayor información que el agente reciba de un estado determinado, más acertadas serán las acciones que pueda tomar en detrimento de una mejor recompensa. Bajo esta presunción nace la idea de emplear un autoencoder LSTM que permita operar con un mayor número de indicadores técnicos. El objetivo de esta arquitectura es entrenar el modelo del autoencoder de forma íntegra y posteriormente, emplear únicamente la parte del encoder para reducir las dimensiones de los datos de entrada, con el fin de no cargar de forma desmedida la dimensionalidad del problema en cuestión. En este sentido, el objetivo es caracterizar mejor los datos de entrada del modelo. Para ello, se construyen los indicadores empleando tres ventanas móviles de diferentes duraciones (5, 20, 60 días) para así poder representar de forma más fidedigna lo que sucede en el corto, medio y largo plazo. Junto con los indicadores ya presentados anteriormente, se añaden también las bandas de Bollingner con el fin de caracterizar la volatilidad del precio de las acciones. Por tanto, se tienen como entrada un total de 64 variables que se pretenden reducir a un espacio latente conformado por únicamente 7 dimensiones. A continuación, se explica de forma detallada todo el procedimiento desarrollado.

En primer lugar, se dividen los datos en un subconjunto de entrenamiento, que incluye alrededor del 70 % de los mismos, y otro de test para validar el método. A continuación, se normalizan los datos de entrada empleando una transformación que escala entre los valores máximos y mínimos de las series temporales de los datos de entrenamiento, y guarda el resultado de los mismos para aplicar la misma transformación al subconjunto de test, pero sin volver a calcular los parámetros de la normalización, para no caer en ningún tipo de sesgo lookahead.

Una vez que se disponen los datos, se procede a la construcción del autoencoder LSTM. Se ha decido implementar el autoencoder con redes recurrentes debido a la naturaleza secuencial de los datos de entrada. Estas son capaces de aprender la compleja dinámica dentro del ordenamiento temporal de las secuencias de entrada, así como de utilizar una memoria interna para recordar o utilizar la información a través de largas secuencias de entrada. El procedimiento interno es muy sencillo, para un conjunto determinado de series de indicadores, se configura el encoder y el decoder para leer la secuencia de entrada, codificarla, decodificarla y recrearla. El rendimiento del modelo se evalúa en función de la capacidad del mismo para recrear la serie de entrada. Una vez que se alcanza un grado de precisión determinado, se desacopla el encoder del modelo y se utiliza para reducir la dimensión de

los datos de entrada. En la Figura 4.6 se adjunta la estructura del modelo descrito, tal y como se puede observar, se trata de una arquitectura simétrica conformada en primer lugar, por el encoder que codifica la serie histórica del conjunto de indicadores, y en segundo lugar, por el decoder que vuelve a recrearla. Ambos tienen dos capas LSTM y una de Dropout para regularizar el modelo y evitar el sobreajuste.

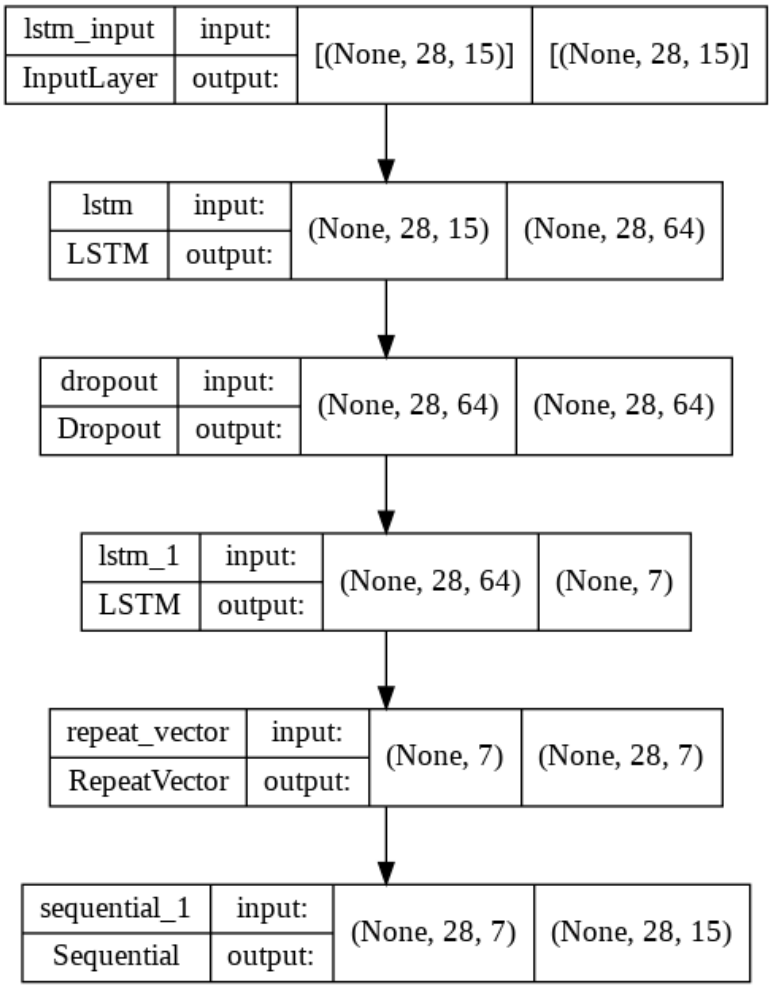


Figura 4.6. *Estructura del autoencoder LSTM*

Con respecto al entrenamiento del modelo, se emplea como función de coste el error absoluto de la diferencia entre la entrada y la predicción (para evitar sobreponderar los outliers) y como optimizador el Adam. Se realiza el entrenamiento por mini-batches durante un total de 200 épocas. Los resultados del modelo se

adjuntan en las Figuras 4.7 y 4.8. Tal y como se puede observar en la Figura 4.8, el número de épocas es el apropiado (incluso sería razonable bajarlo) pues ahí se produce el codo en la curva de validación, y de lo contrario el modelo perdería capacidad de generalización.

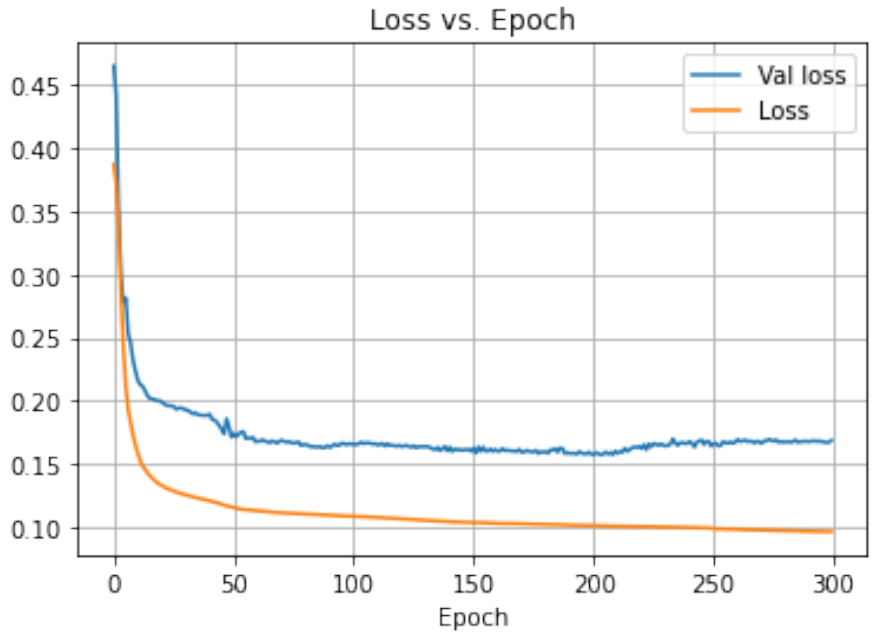


Figura 4.7. *Evolución del loss en el entrenamiento y validación*

Para validar la calidad de reconstrucción del modelo se incluyen las Figuras 4.9 y 4.10 en la que se visualizan, en 6 escenarios diferentes escogidos al azar, cómo se reproducen cualquiera de los 64 indicadores de todas las acciones empleando información combinada de sólo 7 de ellos.

Tal y como se puede observar, los resultados no son del todo satisfactorios ya que las señales de entrada poseen mucho ruido que dificulta la réplica de la misma. Sería interesante aplicar alguna técnica de filtrado, como las basadas en los filtros de Kalman, para reducir las perturbaciones de las entradas. En el capítulo de resultados se valorará el impacto de esta técnica en cuanto a la mejora de rentabilidad que puede ofrecer con respecto al caso base inicial.

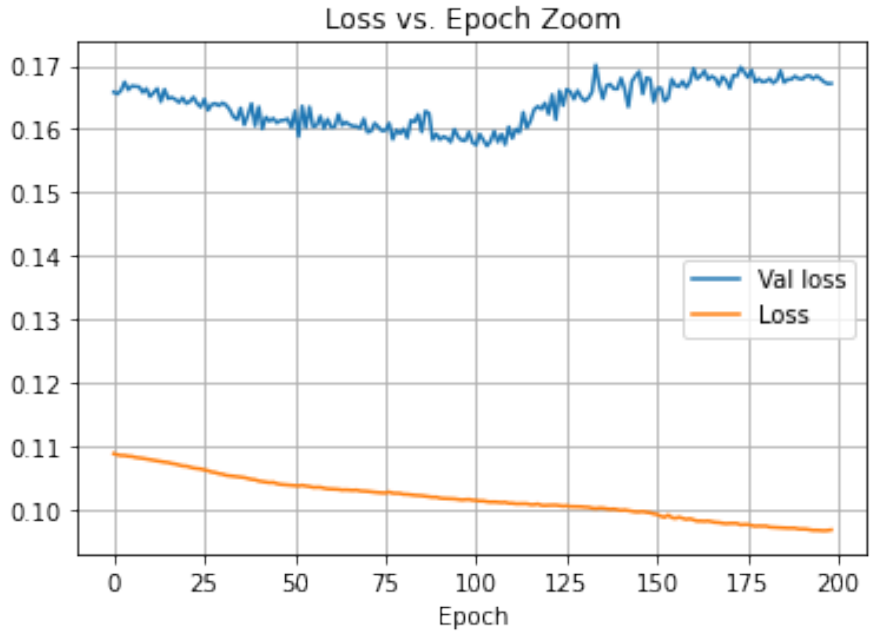


Figura 4.8. *Zoom para validar la convergencia del modelo*

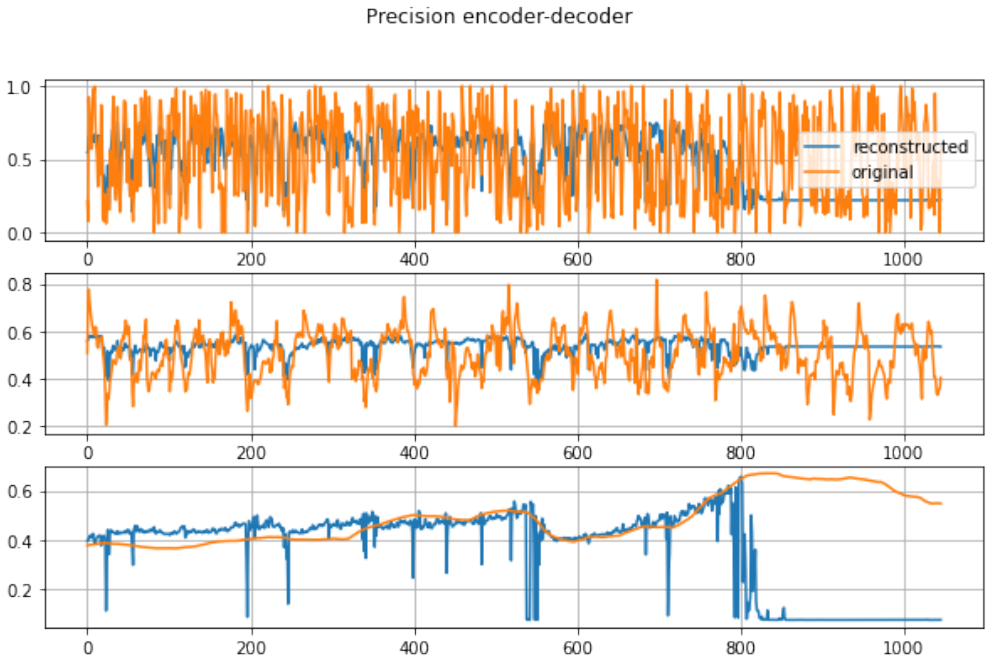


Figura 4.9. *Distintas reconstrucciones del autoencoder (I)*

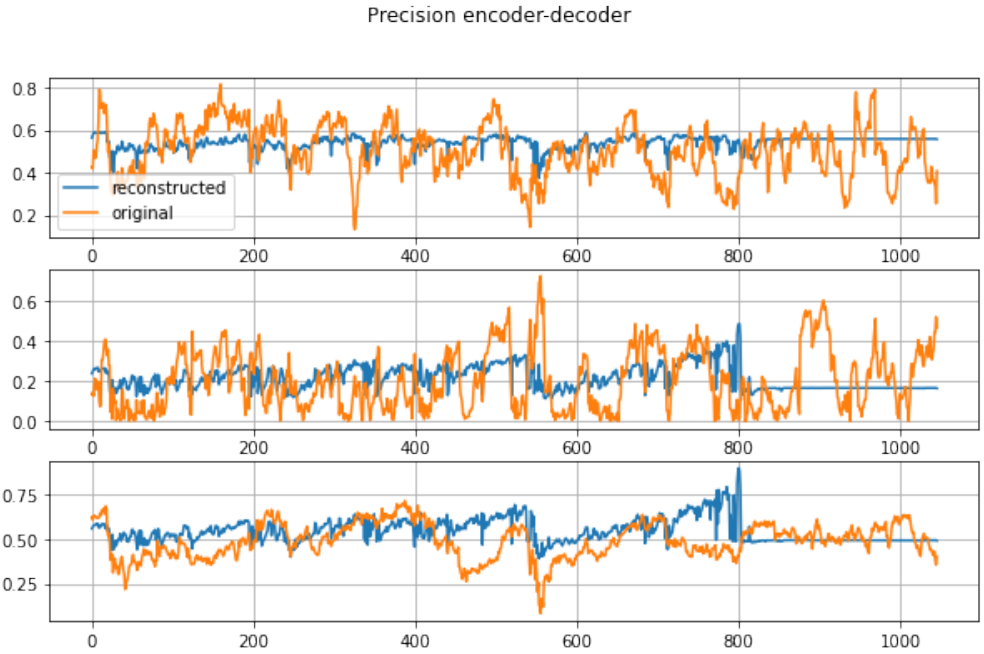


Figura 4.10. *Distintas reconstrucciones del autoencoder (II)*

4.5. Construcción del entorno de DRL

En esta etapa es cuando se adapta el modelo *gym* de OpenAI para al ámbito de la gestión de una cartera de valores. Como se ha enunciado anteriormente, se ha diseñado un simulador muy versátil que permite implementar diferentes tipos de estrategias de *portfolio allocation* según las preferencias de cada usuario. En este sentido, en el capítulo de resultados se analizará el impacto de cada una de las decisiones que el agente humano desee tomar. Se estudiará la influencia del periodo de rebalanceo, del factor de descuento para otorgar más peso a las recompensas futuras frente a las presentes, de la aversión al riesgo mediante el uso del ratio de Sharpe y de las posibles configuraciones de inicialización de los pesos y el impacto en la búsqueda de soluciones.

4.6. Implementación de los algoritmos de DRL

Finalmente, en esta sección se explica de forma detallada el funcionamiento interno de los 5 diferentes algoritmos de DRL empleados.

4.6.1. A2C

En el ámbito del RL el A2C, *Advantage Actor Critic* combina dos tipos de algoritmos de aprendizaje por refuerzo, los basados en políticas y los basados en valores, de forma simultánea. El algoritmo actor-crítico consiste en dos redes (el actor y el crítico) que trabajan juntas para resolver un problema concreto. A alto nivel, la función de ventaja calcula el error de *TD* o el error de predicción del agente. La red de actor elige una acción en cada paso de tiempo y la red de crítico evalúa la calidad o el Q-valor de un estado de entrada dado. A medida que la red crítica aprende qué estados son mejores o peores, el actor utiliza esta información para enseñar al agente a buscar estados buenos y evitar los malos. En esta explicación falta por incluir qué significa el término *Advantage* en este modelo. Para ello, es necesario primero entender cómo se calcula el error. En el aprendizaje temporal diferencial, los agentes aprenden realizando predicciones de las recompensas futuras y ajustando sus acciones conforme al error cometido en la predicción. Para calcular la función de ventaja, primero se tiene que computar el TD objetivo:

$$TD_{target} = R + \gamma\left(S'\right) \tag{4.2}$$

donde $V(S')$ representa la red neuronal crítica computando el valor del siguiente estado S'. Una vez calculado el TD objetivo el error se calcula como la diferencia entre este y el valor de la red neuronal crítica en dicho estado. En A2C la parte de ventaja se puede interpretar también como el error de predicción del agente. De este modo, otorga información sobre si un estado es mejor o no de lo esperado. Por otro lado, el actor se encarga de mapear un estado determinado con una acción devolviendo una distribución de probabilidad para cada posible acción. Por último, el crítico se encarga de mapear cada estado con su Q-valor para representar así la calidad del mismo. Para actualizar los pesos de las redes se optimiza el error cuadrático medio de la función TD previamente descrita. A continuación, se adjunta el pseudocógido del algoritmo diseñado.

Algorithm 1: Advantage Actor Critic (A2C)

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) \Big|_{\theta_k} \hat{A}_t$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

typically via some gradient descent algorithm.
9 : **end for**

4.6.2. PPO

Este tipo de algoritmo se sitúa bajo en la vertiente de métodos de *policy gradient* en los que la política se actualiza de forma explícita, mediante la siguiente función:

$$\nabla J(\theta) = E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right) \hat{A}_t\right] \tag{4.3}$$

Como ya se ha discutido previamente, el principal problema de este tipo de algoritmos es la varianza del gradiente y la idea general es reducirla aplicando la función de ventaja. Posteriormente, se aplica la técnica del ascenso del gradiente para actualizar los parámetros en cada iteración. En el caso del PPO esta optimización se realiza definiendo primero el ratio entre la nueva política y la política pasada según la siguiente relación:

$$r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta old}(a \mid s)} \tag{4.4}$$

Esto permite modificar la función objetivo del método TRPO según la siguiente fórmula:

$$J(\theta)^{TRPO} = E\left[r(\theta)\hat{A}_{\theta old}(s,a)\right] \tag{4.5}$$

Sin añadir restricciones adicionales, esta función objetivo o bien es muy inestable, o converge muy lentamente debido al tamaño de paso usado. En lugar de añadir complicadas limitaciones como la divergencia de KL, PPO impone que el ratio de política $r(\theta)$ debe de estar en un intervalo centrado entre $[1-\epsilon, 1+\epsilon]$. Con esta simplificación se puede reescribir la función anterior como sigue:

$$J^{\text{CLIP}}(\theta) = \mathbb{E}\left[\text{mín}\left(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a)\right)\right] \tag{4.6}$$

donde la función *clip* es la que se encarga precisamente de realizar el truncamiento en el ratio de la política. A continuación, se adjunta el pseudocódigo del algoritmo implementado.

Algorithm 2: Proximal Policy Optimization (PPO)

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \right.$$
$$\left. g\left(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)\right) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

8: **end for**

4.6.3. DDPG

Deep Deterministic Policy Gradient (DDPG) es un algoritmo que aprende simultáneamente una función Q y una política. Utiliza datos fuera de la política y la ecuación de Bellman para aprender la función Q, y utiliza la función Q para aprender la política.

Este enfoque está estrechamente relacionado con el Q-learning, y está motivado inicialmente de la misma forma: si se conoce la función óptima de acción-valor $Q^*(s,a)$, entonces en cualquier estado dado, la acción óptima $a^*(s)$ se puede encontrar resolviendo la siguiente ecuación:

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{4.7}$$

El DDPG intercala el aprendizaje de un aproximador de $Q^*(s,a)$ con el aprendizaje de un aproximador de $a^*(s)$, y lo hace de una manera que está específicamente adaptada para entornos con espacios de acción continuos. Para ello, modifica la forma en la que se calcula el máximo sobre las acciones en $\max_a Q^*(s,a)$.

Cuando hay un número finito de acciones discretas, el máximo no supone ningún problema, porque se pueden calcular los valores Q de cada acción por separado y compararlos directamente (esto también da inmediatamente la acción que maximiza el valor Q.) Pero cuando el espacio de acciones es continuo, no se puede evaluar exhaustivamente el espacio, y resolver el problema de optimización no es nada trivial. El uso de un algoritmo de optimización normal haría que el cálculo de $\max_a Q^*(s,a)$ fuera una subrutina dolorosamente cara. Y dado que tendría que ejecutarse cada vez que el agente quiera realizar una acción en el entorno, esto es inaceptable.

Como el espacio de acción es continuo, se supone que la función $Q^*(s,a)$ es diferenciable con respecto al argumento de la acción. Esto permite establecer una regla de aprendizaje eficiente, basada en el gradiente, para una política $\mu(s)$ que explota este hecho. Entonces, en lugar de ejecutar una subrutina de optimización costosa cada vez que se calcule $\max_a Q^*(s,a)$, se puede aproximar con $\max_a Q(s,a) \approx Q(s,\mu(s))$. Para entender con mayor profundidad la parte matemática de cada una de las partes del DDPG consultar el Apéndice B.

Algorithm 3: Deep Deterministic Policy Gradient (DDPG)

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta, \phi_{\text{targ}} \leftarrow \phi$
repeat
    Observe state s and select action $a = \text{clip}\left(\mu_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}}\right)$, where $\epsilon \sim \mathcal{N}$
Execute a in the environment
Observe next state s', reward r, and done signal d to indicate whether s' is terminal
Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
If s' is terminal, reset environment state.
if it's time to update **then**
    for however many updates **do**
        Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
        Compute targets
$$y\left(r, s', d\right) = r + \gamma(1-d) Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}\left(s'\right)\right)$$
        Update Q-function by one step of gradient descent using
$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s,a) - y\left(r, s', d\right)\right)^2$$
        Update policy by one step of gradient ascent using
$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi\left(s, \mu_\theta(s)\right)$$
        Update target networks with
$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1-\rho)\theta$$
    end for
end if
until convergence

4.6.4. SAC

El algoritmo SAC, *Soft Actor Critic* optimza una función de política estocástica de forma offline, estableciendo un puente entre las políticas estocásticas y el enfoque impregnado por los métodos derivados del algoritmo DDPG. Incorpora también el ajuste del recorte de la doble función Q y, al ser una método estocástico, también se beneficia del suavizado de la función política. La característica fundamental de este tipo de algoritmos es la regularización de la entropía. En este sentido, la política se entrena para maximizar el *trade-off* entre la recompensa esperada y la entropía (como medida de la aleatoriedad de la misma). Esto es una forma de representar el dilema entre la exploración y explotación presentes en todos los agentes de DRL.

Para explicar el algoritmo SAC es necesario introducir el concepto de la regularización de entropía que se puede encontrar en el Apéndice A. Dicho método aprende de forma concurrente una política π_θ y dos funciones Q, Q_{ϕ_1}, Q_{ϕ_2}. Para aprender dichas funciones, se emplea un procedimiento similar al discutido en la sección del algoritmo TD3, pero aplicando tres diferencias significativas:

1. La función objetivo también incluye el término de regularización de la entropía.

2. Las acciones del siguiente estado en la función objetivo proceden de la política actual, en lugar de la política objetivo.

3. No se aplica de forma explícita ningún suavizado en la función política, aunque, como ya se ha comentado anteriormente, el hecho de que sea estocástico produce un efecto ligeramente parecido.

Para entrenar cada una de las Q funciones se emplea el error cuadrático medio de la ecuación de Bellman, quedando la función de coste como sigue:

$$L\left(\phi_i, \mathcal{D}\right) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_{\phi_i}(s,a) - y\left(r,s',d\right)\right)^2\right] \tag{4.8}$$

donde la función objetivo viene representada por la siguiente ecuación:

$$y\left(r,s',d\right) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ}},j}\left(s',\tilde{a}'\right) - \alpha\log\pi_\theta\left(\tilde{a}' \mid s'\right)\right), \quad \tilde{a}' \sim \pi_\theta\left(\cdot \mid s'\right) \tag{4.9}$$

Para aprender la política, en cada estado se trata de maximizar el balance entre la futura recompensa y entropía a través de esta relación:

$$\begin{aligned} V^\pi(s) &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a)\right] + \alpha H(\pi(\cdot \mid s)) \\ &= \underset{a\sim\pi}{\mathrm{E}}\left[Q^\pi(s,a) - \alpha\log\pi(a \mid s)\right]. \end{aligned} \tag{4.10}$$

De esta forma, se optimiza la política haciendo uso del truco de la reparametrización, en el que una muestra obtenida a partir de $\pi_\theta(\cdot \mid s)$ computa una función determinista del estado, los parámetros de la política y de un ruido independiente. Esto permite reescribir la expectación sobre las acciones en la expectación sobre el ruido de la siguiente forma:

$$\underset{a\sim\pi_\theta}{\mathrm{E}}\left[Q^{\pi_\theta}(s,a)-\alpha\log\pi_\theta(a\mid s)\right]=\underset{\xi\sim\mathcal{N}}{\mathrm{E}}\left[Q^{\pi_\theta}\left(s,\tilde{a}_\theta(s,\xi)\right)-\alpha\log\pi_\theta\left(\tilde{a}_\theta(s,\xi)\mid s\right)\right] \tag{4.11}$$

Finalmente, es necesario sustituir el valor de Q_{ϕ_θ} con una de las funciones aproximadoras, empleando la menor de las dos aproximaciones ($\text{mín}_{j=1,2}\, Q_{\phi_j}$, a diferencia del TD3 que siempre emplea Q_{ϕ_1}. De este modo, la política se optimziza de acuerdo con la siguiente fórmula:

$$\underset{\theta}{\text{máx}}\ \underset{\substack{s\sim\mathcal{D}\\ \xi\sim\mathcal{N}}}{\mathrm{E}}\left[\min_{j=1,2} Q_{\phi_j}\left(s,\tilde{a}_\theta(s,\xi)\right)-\alpha\log\pi_\theta\left(\tilde{a}_\theta(s,\xi)\mid s\right)\right] \tag{4.12}$$

A continuación, se adjunta el pseudocódigo del algoritmo desarrollado.

Algorithm 4: Soft Actor Critic (SAC)

Input: initial policy parameters θ_1, θ_2, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ},1} \leftarrow \theta, \phi_{\text{targ},1} \leftarrow \phi$, $\theta_{\text{targ},2} \leftarrow \theta, \phi_{\text{targ},2} \leftarrow \phi$
repeat
    Observe state s and select action $a \sim \pi_\theta(\cdot \mid s)$
Execute a in the environment
Observe next state s', reward r, and done signal d to indicate whether s' is terminal
Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
If s' is terminal, reset environment state.
if it's time to update **then**
    for however many updates **do**
        Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
        Compute targets for Q functions
$$y(r, s', d) = r + \gamma(1 - d)\left(\text{mín}_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' \mid s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot \mid s')$$
        Update Q-function by one step of gradient descent using
$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \quad \text{for } i = 1, 2$$
        Update policy by one step of gradient ascent using
$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\text{mín}_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s) \mid s)\right)$$
        where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot \mid s)$ which is differentiable wrt θ via the reparametrization trick.
        Update target networks with
$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \quad \text{for } i = 1, 2$$
    end for
end if
until convergence

4.6.5. TD3

Aunque el algoritmo DDPG puede alcanzar un gran rendimiento, es muy sensible a los hiperparámetros empleados. Un fallo común en este tipo de algoritmos es que la función Q aprendida empieza a sobreestimar los Q-valores, lo que conduce al problema de la explosión del gradiente. Para hacer frente a este problema el algoritmo TD3 introduce tres técnicas claves:

1. Clipped Double-Q Learning: el TD3 aprende dos funciones Q en lugar de sólo una y emplea la que proporcione Q-valores más pequeños para construir

los valores objetivos de la ecuación de Bellman.

2. Delayed Policy Updates: dicho algoritmo actualiza las políticas (y las redes neuronales) con menor frecuencia que la función Q. Normalmente dicha actualización tiene un perido el doble de lento.

3. Target Policy Smoothing: el TD3 añade ruido a la acción objetivo para que sea más dificl que la política explote los Q-valores.

Por lo tanto, como se ha mencionado anteriormente, el algoritmo aprende de forma simultánea dos Q funciones y las optimiza mediante el método de Bellman. Durante el entrenamiento, las acciones empleadas en la función objetivo de Q-learning se basan en la política objetivo $\mu\theta_{targ}$ pero añadiendo el ruido que ya se ha comentado anteriormente. Tras ello, se escala el valor de las acciones para que no se infrinjan los límites establecidos ($a_{Low} \leq a \leq a_{High}$), quedando representadas dichas acciones por la siguiente ecuación:

$$a'\left(s'\right) = \text{clip}\left(\mu_{\theta_{\text{targ}}}\left(s'\right) + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{4.13}$$

De este modo el suavizado de la función política objetivo se puede visualizar como un regularizador para dicho algoritmo. A continuación, se emplea el primer ajuste mencionado anteriormente aplicando la siguiente fórmula para cada una de las Q funciones y escogiendo la que resulte en un menor.

$$y\left(r, s', d\right) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}\left(s', a'\left(s'\right)\right) \tag{4.14}$$

Posteriormente, ambas funciones aprenden mediante el retroceso a su valor objetivo.

$$\begin{aligned} L\left(\phi_1, \mathcal{D}\right) &= \underset{(s,a,r,s',d)\sim\mathcal{D}}{\text{E}}\left[\left(Q_{\phi_1}(s,a) - y\left(r, s', d\right)\right)^2\right], \\ L\left(\phi_2, \mathcal{D}\right) &= \underset{(s,a,r,s',d)\sim\mathcal{D}}{\text{E}}\left[\left(Q_{\phi_2}(s,a) - y\left(r, s', d\right)\right)^2\right]. \end{aligned} \tag{4.15}$$

Al usar el menor de los valores de Q se evita la sobreestimación de los Q-valores y se reduce la inestabilidad del método en comparación con la presente en el DDPG. Finalmente, se aprende la política maximizando Q_{ϕ_1}.

$$\max_{\theta} \text{E}_{s\sim\mathcal{D}}\left[Q_{\phi_1}\left(s, \mu_\theta(s)\right)\right] \tag{4.16}$$

A continuación, se adjunta el pseudocódigo del algoritmo descrito.

Algorithm 5: Twin Delayed DDPG (TD3)

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta, \phi_{\text{targ, 1}} \leftarrow \phi_1, \phi_{\text{targ,2}} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}\left(\mu_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}}\right)$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions
$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$
      Compute targets
$$y(r, s', d) = r + \gamma(1 - d)\,\text{mín}_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$
      Update Q-functions by one step of gradient descent using
$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \quad \text{for } i = 1, 2$$
      if j mód policy_delay $= 0$ **then**
        Update policy by one step of gradient ascent using
$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$
        Update target networks with
$$\begin{aligned} \phi_{\text{targ},i} &\leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \quad \text{for } i = 1, 2 \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta \end{aligned}$$
      end if
    end for
  end if
until convergence

Finalmente, se incluyen en la Tabla 4.1 los valores de los hiperparámetros con los que se han entrenado cada uno de los modelos descritos anteriormente.

	A2C	**PPO**	**DDPG**	**SAC**	**TD3**
N steps	5	2048	*N.A*	2048	2048
Ent coefficient	0,005	0,005	0,507	*N.A*	*N.A*
Batch size	*N.A.* [2]	*N.A*	128	128	100
Buffer size	*N.A*	*N.A*	100000	100000	100000
Learning rate	0,0002	0,0001	0,001	0,0003	0,001
Learning starts	*N.A*	*N.A*	*N.A*	100	*N.A*

Tabla 4.1. *Hiperparámetros de cada uno de los modelos*

Capítulo 5

Resultados

El objetivo fundamental de este capítulo es exponer el procedimiento llevado a cabo para contrastar el rendimiento de la estrategia de inversión diseñada sometiéndola frente a diferentes escenarios para así poder evaluar sus principales limitaciones. Como ya se ha comentado anteriormente, el sistema implementado posee gran flexibilidad y multitud de parámetros ajustables según las preferencias del gestor. Por lo tanto, la principal motivación de esta sección es entender el efecto de cada uno de ellos sobre la rentabilidad final de la cartera y evaluar el rendimiento del portfolio frente a los benchmarks establecidos. Para ser rigurosos en la metodología seguida se expondrá de forma detalla el análisis secuencial llevado a cabo en los sucesivos apartados.

- En primer lugar, se tratará de encontrar el modelo que mejor se adapte a las funcionalidades del problema. Para ello, se simularán diferentes entornos virtuales con distintos periodos de rebalanceo de la cartera. Se elegirá, para proseguir en el análisis pormenorizado, la combinación de modelo y rebalanceo que otorguen mayor rentabilidad.
- En segundo lugar, la sensibilidad a la función de recompensa, así como la inicialización de los pesos.
- En tercer lugar, se valorará la eficacia de adoptar la arquitectura del autoencoder LSTM para incluir un mayor número de indicadores técnicos y por lo tanto, enriquecer la fuente de datos que el agente recibe en el entorno.
- En cuarto lugar, se analizará si resulta conveniente aplicar un filtro previo que separe las acciones según su volatilidad y que por lo tanto, diseñe agentes de RL que se puedan adaptar con mayor facilidad a su entorno.

- En quinto lugar, se medirá la influencia del sector en el grado de exposición de la cartera según el mismo.
- En sexto lugar, se realizará una validación cruzada del mejor modelo alcanzado hasta el momento para valorar la robustez del método frente a diferentes situaciones del mercado (tendencias y volatilidad).
- En último lugar, se comparará el rendimiento del algoritmo frente a tres estrategias empleadas tradicionalmente en el sector: comprar y mantener un ETF del índice, un portfolio equiponderado y un portfolio construido según Markowitz.

Como se puede observar, se trata de un análisis muy exhaustivo que permite entender mejor cómo se comporta el modelo frente a diferentes estímulos. Para poder llevar a cabo todas las ejecuciones, resulta crucial disponer de una herramienta de backtesting sólida que permita simular cómo se comportaría el modelo en el pasado, siendo conscientes de que beneficios pasados no garantizan beneficios futuros. Para ello, se emplean las funciones definidas en la librería de FinRL y se acompaña de análisis estadístico y gráficos ilustrativos que permitan entender el comportamiento del modelo. Finalmente, para poder llevarlo a cabo en la práctica, se dividirá el conjunto total de los datos en dos particiones: del 1 de enero de 2008 hasta el 1 de enero de 2018 se utilizará como fuente de entrenamiento y desde esa fecha hasta el 28 de febrero de 2022 como periodo de validación. Todos los resultados que se presenten son relativos al segundo subconjunto, es decir, se llevara a cabo un *out-of-sample* análisis.

5.1. Escenario I: efecto del periodo de rebalanceo y obtención del mejor modelo

En este primer apartado, se evaluará el efecto de emplear diferentes períodos de rebalanceo para cada uno de los modelos descritos en el capítulo anterior. El entorno disponible en FinRL no permite realizar este análisis y como tampoco tienen en cuenta los costes de transacción, realmente el sistema diseñado se aleja bastante de la realidad bursátil. Por ello, el objetivo fundamental de este apartado es que la brecha entre el agente virtual y cualquier operador humano sea la mínima posible. Se partirá de un capital inicial de $1M.

Para evaluar el rendimiento de cada uno de los modelos se emplearán sucesivos indicadores estadísticos que permiten valorar cómo de eficiente es el agente gestionando las carteras. Dicho análisis se repetirá en cada uno de los 4 escenarios simulados: diario, semanal, mensual y trimestral. Además, se comparará el rendimiento del mejor modelo frente al del índice.

5.1.1. Rebalanceo diario

En primer lugar, se adjunta en la Figura 5.1 la rentabilidad acumulada por cada uno de los modelos en el periodo de trading.

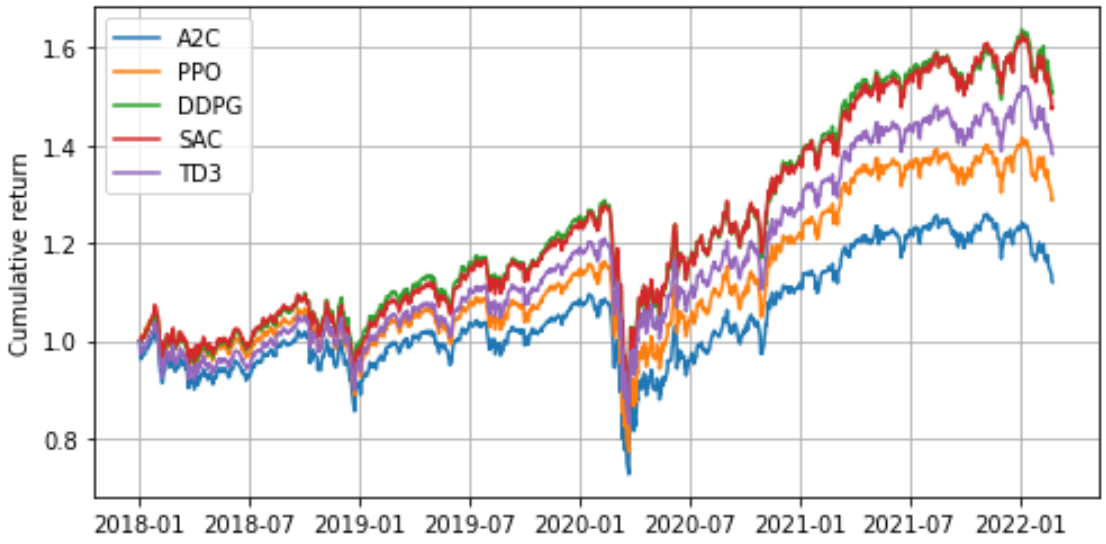


Figura 5.1. *Rentabilidad acumulada por cada uno de los modelos con rebalanceo diario*

Para un análisis más profundo, se adjunta en la Tabla 5.1 las estadísticas del portfolio para cada uno de los modelos de RL. Como se puede observar, el mejor modelo es el DDPG no sólo en cuanto a la maximización de la rentabilidad, sino también con respecto a la aversión al riesgo. [1]

Sin embargo, si se analiza mejor lo que realiza el agente a lo largo del periodo de validación, se puede observar en la Figura 5.2 cómo únicamente realiza dos

[1]Para una explicación más detallada del significado de cada uno de los ratios consultar el Apéndice C

	A2C	PPO	DDPG	SAC	TD3
Annual return	0,028	0,063	0,104	0,099	0,081
Cumulative returns	0,120	0,289	0,507	0,477	0,382
Annual volatility	0,205	0,211	0,212	0,209	0,203
Sharpe ratio	0,236	0,396	0,573	0,554	0,486
Calmar ratio	0,082	0,188	0,310	0,301	0,258
Stability	0,618	0,712	0,854	0,857	0,841
Max drawdown	−0,336	−0,335	−0,336	−0,327	−0,314
Omega ratio	1,050	1,086	1,126	1,122	1,105
Sortino ratio	0,324	0,549	0,804	0,768	0,678
Skew	−0,379	−0,364	−0,291	−0,561	−0,269
Kurtosis	16,993	18,360	17,925	19,044	16,709
Tail ratio	0,791	0,847	0,900	0,840	0,857
Daily value at risk	−0,026	−0,026	−0,026	−0,026	−0,025

Tabla 5.1. *Estadísticas del Porfolio para todos DRL- Rebalanceo diario*

operaciones a lo largo de los 4 años, lo cual carece completamente de sentido en el problema en cuestión. Esto se puede deber a que el agente aprende que para no tener que hacer frente a los costes de transacción, lo mejor es simplemente implementar a una estrategia de comprar y mantener a largo plazo.

Si en lugar de emplear el modelo DDPG se utiliza el segundo mejor, el *Soft Critic Agent*, el comportamiento ya es más generalizable frente a situaciones futuras, como se puede observar en la Figura 5.3. Por lo tanto, con este ejemplo se pone claramente de manifiesto que el modelo que sea más rentable no es necesariamente el mejor, porque si su rendimiento no es sostenible en el tiempo carece completamente de sentido.

Como se ha indicado anteriormente, se procede a realizar un análisis más exhaustivo del comportamiento del modelo SAC. En primer lugar, se adjunta en la Figura 5.4 los beneficios acumulados del modelo SAC junto con el rendimiento del índice. Como se puede observar, el modelo bate al mercado en el periodo simulado.

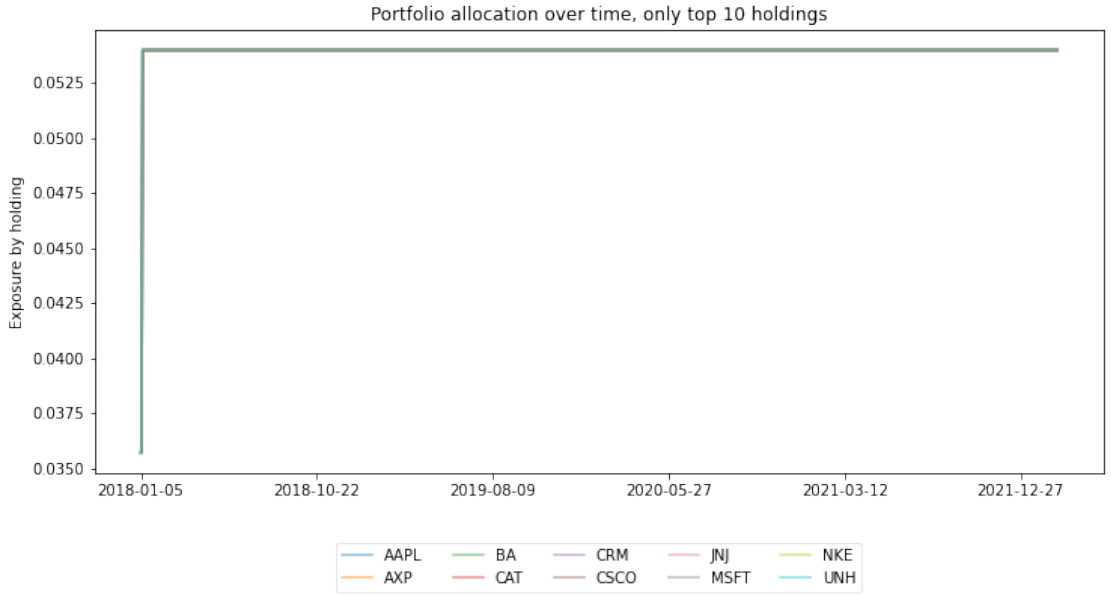


Figura 5.2. *Asignación de capital del modelo DDPG con un rebalanceo diario*

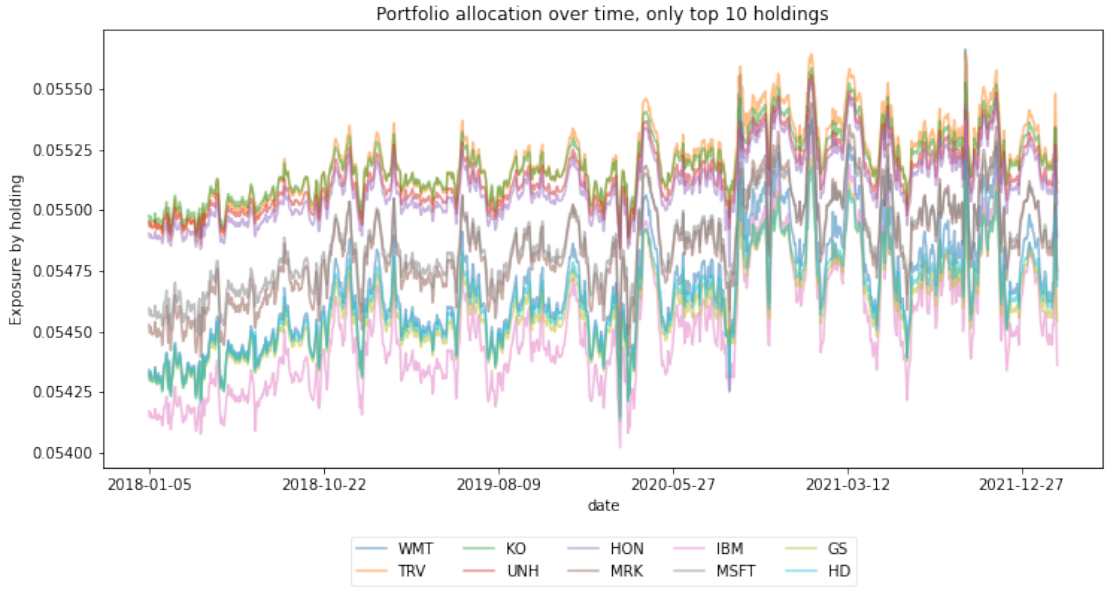


Figura 5.3. *Asignación de capital del modelo SAC con un rebalanceo diario*

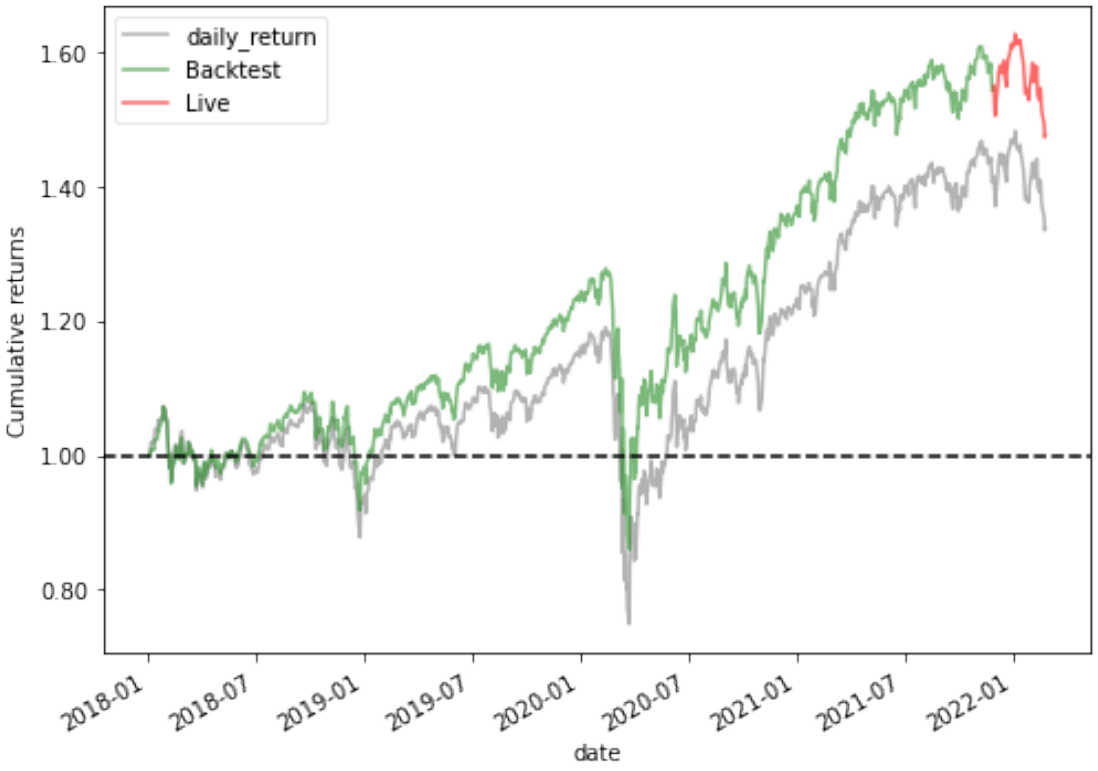


Figura 5.4. *Comparación de los rendimientos del SAC y del índice para el rebalanceo diario*

Además, no sólo se comporta mejor que el mercado en términos de la rentabilidad acumulada, sino que, como se pueden comparar mediante las Tablas 5.2 y 5.3, pierde menos dinero en periodos bajistas (32.72 % frente a 37.09 %) y se recupera antes de las mismas, concretamente casi dos meses y medio antes.

Periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	32,72	2020−02−12	2020−03−23	2020−09−02	146
1	16,07	2018−09−21	2018−03−23	2019−03−21	130
2	11,13	2018−01−26	2018−12−24	2018−09−29	154
3	9,43	2022−01−04	2022−02−23	*NaT*	*NaN*
4	8,08	2020−09−02	2020−10−28	2020−11−09	49

Tabla 5.2. *Worst down periods for SAC model- Rebalanceo diario*

Periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	37,09	2020−02−12	2020−03−23	2020−11−16	199
1	18,77	2018−10−03	2018−12−24	2019−07−03	196
2	11,58	2018−01−26	2018−03−23	2018−09−20	170
3	9,97	2022−01−04	2022−02−23	*NaT*	*NaN*
4	6,87	2019−07−15	2019−08−14	2019−11−04	81

Tabla 5.3. *Worst down periods for DJI- Rebalanceo diario*

Para representar dicho fenómeno resulta muy interesante presentar los gráficos de las Figuras 5.5 y 5.6. El primero de ellos muestra los rendimientos acumulados del modelo señalando los 5 periodos de máximas caídas. Tal y como se puede observar, el modelo es relativamente estable en cuanto a la conquista de beneficios. Por el contrario, el segundo se centra únicamente en las pérdidas y muestra el tiempo que tardó el valor de la cartera en recuperar el máximo anterior, después de sufrir una pérdida. Este gráfico permite distinguir fácilmente entre periodos de pérdidas normales y prolongados. En este caso, salvo la drástica caída provocada por el COVID-19, todos los niveles están razonablemente dentro de los límites habituales.

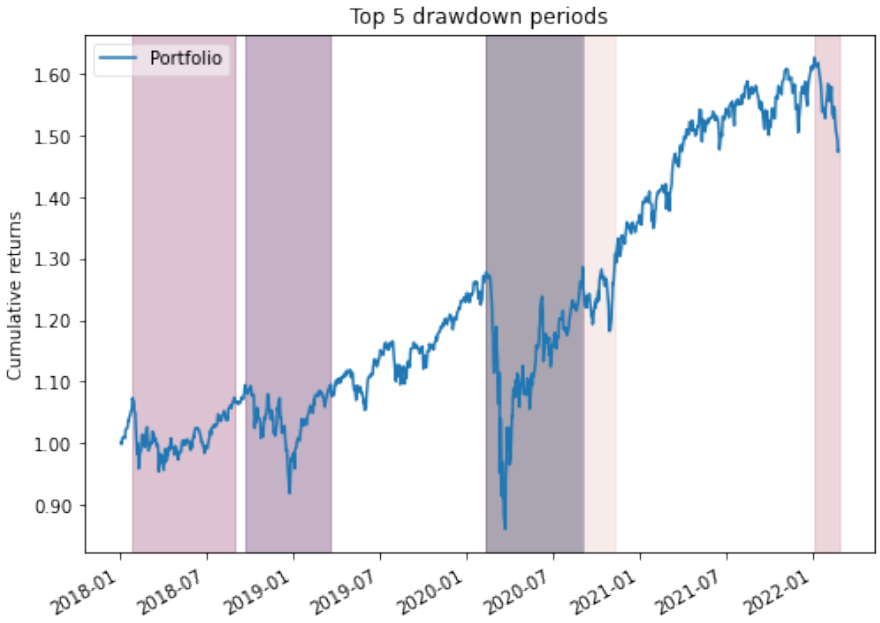


Figura 5.5. *Máximas caídas del rendimiento con el SAC y rebalanceo diario*

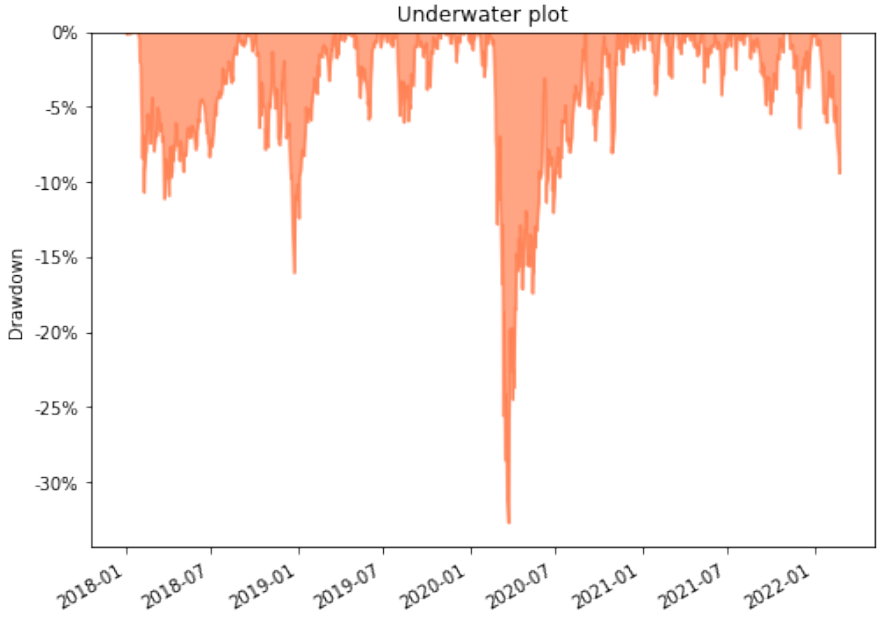


Figura 5.6. *Gráfico de underwater para el rebalanceo diario*

5.1.2. Rebalanceo semanal

En primer lugar, se adjunta en la Figura 5.7 la rentabilidad acumulada por cada uno de los modelos en el periodo de trading.

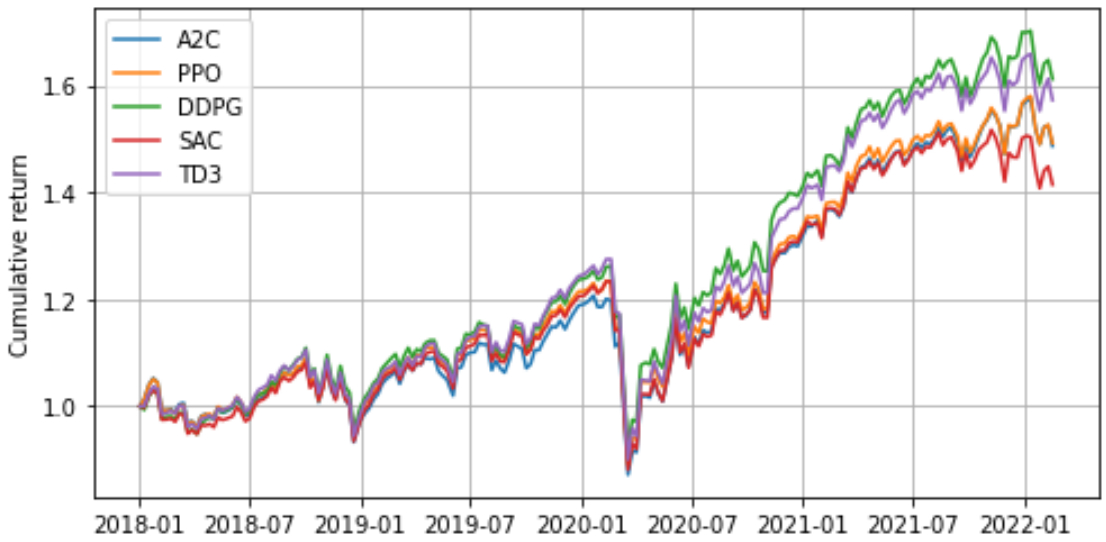


Figura 5.7. *Rentabilidad acumulada por cada uno de los modelos con rebalanceo semanal*

Se procede a realizar un análisis muy análogo que el del apartado anterior. En la Tabla 5.4 se adjuntan los ratios de cada uno de los portfolios según el modelo implementado. De nuevo, el mejor modelo es el DDPG pero le sucede el mismo problema que en el apartado anterior, al igual que le ocurre al TD3. Por lo tanto, se opta por escoger el algoritmo PPO para realizar el análisis.

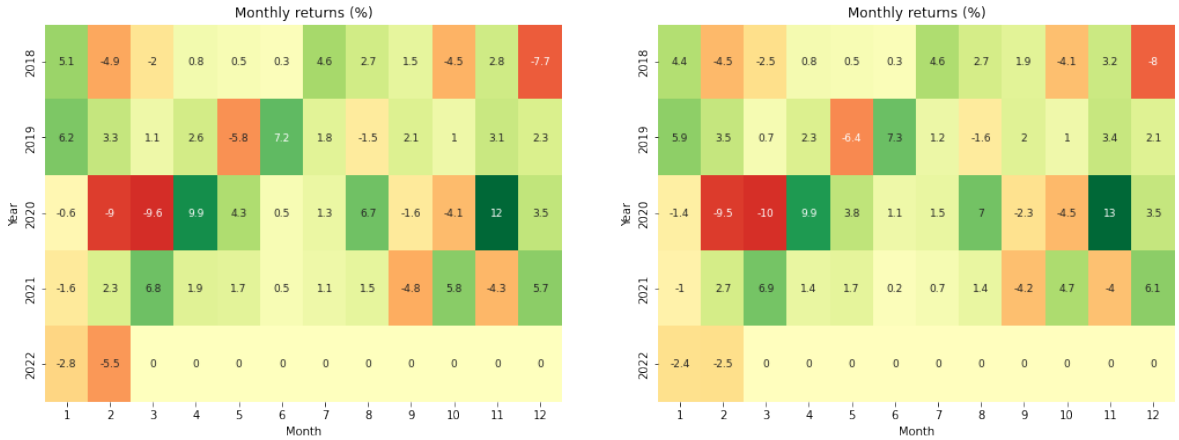
Se observa como en este caso como tanto el beneficio acumulado, como el ratio de Sharpe son prácticamente iguales que el del escenario anterior. Si se analizan los valores de la Tabla 5.5, se puede ver como, aunque el período más severo se comporta peor que el SAC del escenario anterior, en general el beneficio acumulado es más resiliente a las caídas, siendo, sin embargo, los períodos de recuperación prácticamente similares.

En línea con lo anterior, si se comparan, en la Figura 5.8, el *heatmap* de los rendimientos mensuales en ambos casos se puede notar como en la mayoría de casillas señaladas en rojo los rendimientos son superiores en el caso del rebalanceo

	A2C	PPO	DDPG	SAC	TD3
Annual return	0,097	0,098	0,116	0,083	0,111
Cumulative returns	0,468	0,473	0,577	0,390	0,550
Annual volatility	0,211	0,212	0,214	0,213	0,219
Sharpe ratio	0,544	0,547	0,619	0,479	0,593
Calmar ratio	0,295	0,293	0,353	0,239	0,321
Stability	0,790	0,814	0,855	0,795	0,826
Max drawdown	−0,329	−0,333	−0,329	−0,345	−0,347
Omega ratio	1,119	1,121	1,136	1,104	1,130
Sortino ratio	0,757	0,763	0,869	0,668	0,829
Skew	−0,414	−0,365	−0,312	−0,333	−0,338
Kurtosis	17,763	18,238	16,805	17,207	17,140
Tail ratio	0,819	0,862	0,931	0,875	0,909
Daily value at risk	−0,026	−0,026	−0,026	−0,026	−0,027

Tabla 5.4. *Estadísticas del Porfolio para todos DRL- Rebalanceo semanal*

semanal. De este modo, parece razonable aumentar el periodo de rebalanceo.



(a) *Rebalanceo diario* (b) *Rebalanceo semanal*

Figura 5.8. *Mapa de calor de los rendimientos del modelo*

Periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	33,33	2020−02−12	2020−03−23	2020−11−09	194
1	15,91*l*	2018−10−03	2018−12−24	2019−04−21	129
2	11,08	2018−01−26	2018−03−23	2018−09−13	165
3	8,74	2022−01−04	2022−02−23	*NaT*	*NaN*
4	6,64	2021−11−08	2021−12−01	2021−12−27	36

Tabla 5.5. *Worst down periods for PPO model- Rebalanceo semanal*

5.1.3. Rebalanceo mensual

En primer lugar, se adjunta en la Figura 5.9 la rentabilidad acumulada por cada uno de los modelos en el periodo de trading.

Figura 5.9. *Rentabilidad acumulada por cada uno de los modelos con rebalanceo mensual*

En este apartado se abordarán los resultados obtenidos si los pesos de la cartera se actualizan cada mes. La metodología empleado para ello es sustancialmente análoga a los otros dos. En la Tabla 5.6 se incluye los resultados estadísticos de cada uno de los modelos. Tal y como se puede observar, el mejor modelo sigue siendo el DDPG, seguido del TD3 y del PPO. De nuevo, en el caso de los dos primeros ocurre el mismo fenómeno que en las situaciones anteriores, por lo que,

se opta por analizar el rendimiento del algoritmo PPO.

	A2C	PPO	DDPG	SAC	TD3
Annual return	0,064	0,105	0,110	0,083	0,117
Cumulative returns	0,291	0,516	0,540	0,393	0,581
Annual volatility	0,211	0,212	0,215	0,215	0,213
Sharpe ratio	0,398	0,580	0,591	0,480	0,626
Calmar ratio	0,192	0,316	0,322	0,243	0,354
Stability	0,619	0,830	0,823	0,793	0,862
Max drawdown	−0,330	−0,333	−0,340	−0,342	−0,330
Omega ratio	1,085	1,128	1,131	1,105	1,138
Sortino ratio	0,553	0,810	0,826	0,667	0,876
Skew	−0,375	−0,367	−0,434	−0,375	−0,339
Kurtosis	16,912	18,243	19,302	17,982	17,182
Tail ratio	0,851	0,862	0,908	0,896	0,860
Daily value at risk	−0,026	−0,026	−0,027	−0,027	−0,026

Tabla 5.6. *Estadísticas del Porfolio para todos DRL- Rebalanceo mensual*

En la Figura 5.10 se muestra un gráfico de barras con las principales métricas para gestionar una cartera. Tal y como se puede observar, la discrepancia en el ratio de Sharpe viene dado por la diferencia en las rentabilidades y no en la volatilidad de los algoritmos.

A continuación, se analiza en la Tabla 5.7 los periodos consecutivos en negativo. Se observan que son prácticamente iguales que los anteriores.

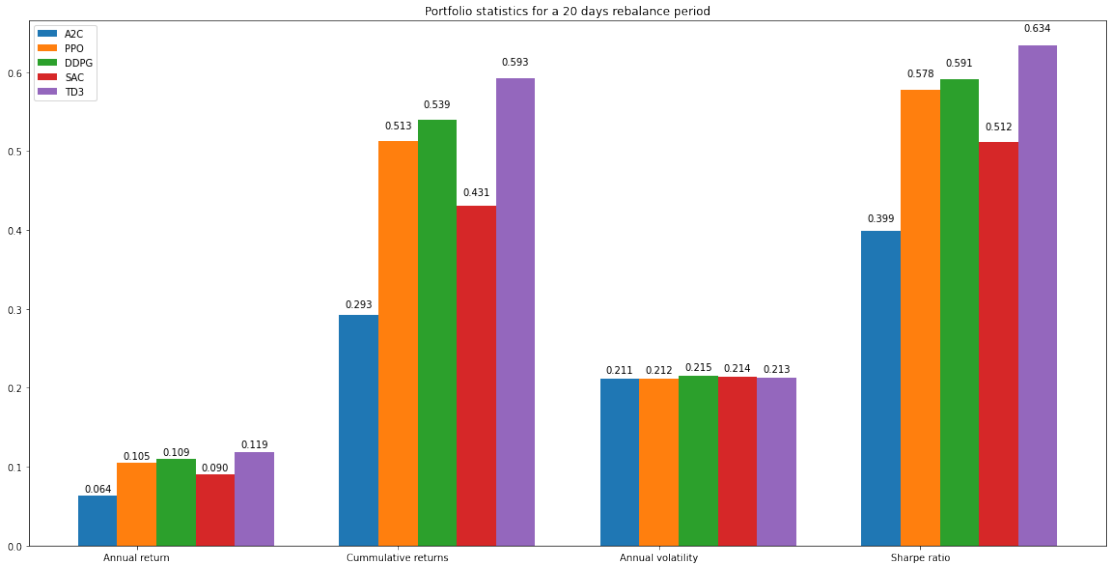


Figura 5.10. *Comparación de las estadísticas del portfolio con un rebalanceo mensual*

Periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	33,31	2020−02−12	2020−03−23	2020−09−02	146
1	15,78	2018−10−03	2018−12−24	2019−03−21	122
2	10,08	2018−01−26	2018−03−23	2018−08−28	153
3	8,72	2022−01−04	2022−02−23	*NaT*	*NaN*
4	8,62	2020−09−02	2020−10−28	2020−11−09	49

Tabla 5.7. *Worst down periods for PPO model- Rebalanceo mensual*

5.1.4. Rebalanceo trimestral

En primer lugar, se adjunta en la Figura 5.11 la rentabilidad acumulada por cada uno de los modelos en el periodo de trading.

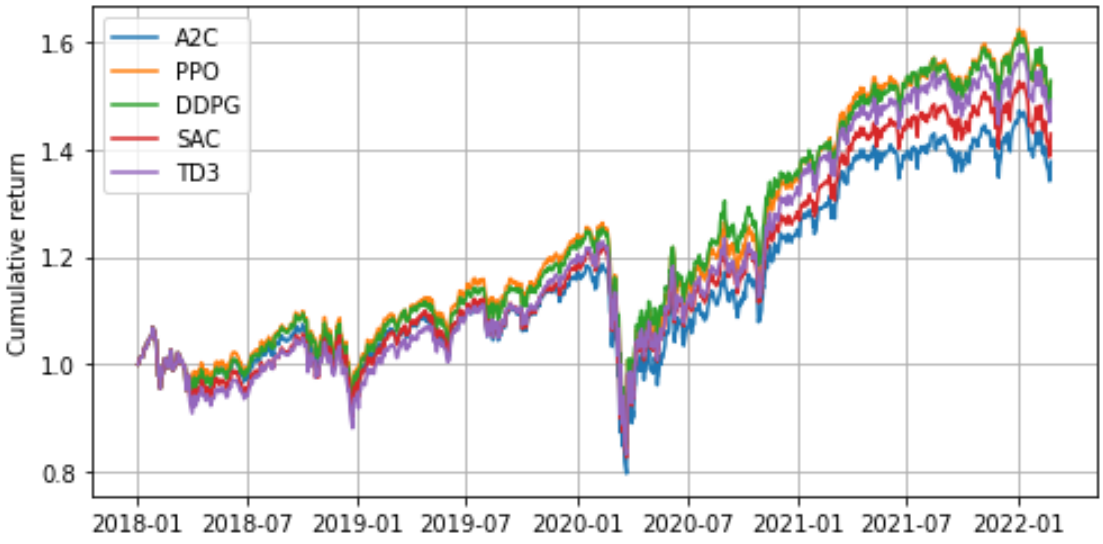


Figura 5.11. *Rentabilidad acumulada por cada uno de los modelos con rebalanceo trimestral*

A continuación, se adjunta en la Tabla 5.8 las métricas para medir el rendimiento del portfolio según el algoritmo empleado. De nuevo, el algoritmo PPO es el que mejor se adapta a las diferentes situaciones del mercado y los resultados obtenidos son muy similares a los analizados en los escenarios anteriores.

No parece muy intuitivo que el factor de rebalanceo no tenga un gran impacto en el rendimiento de los modelos. Por eso, hay que analizar con más detalle cómo asigna capital el agente de DRL a lo largo del tiempo. Tal y como se puede observar en la Figura 5.12 el principal problema es que los pesos están prácticamente equiponderados y esto es lo que origina que el método sea prácticamente invariable a la frecuencia de adaptación del mismo. Esto se puede deber al método que se emplea para la inicialización (uniforme) o a los elevados costes de transacción. Por lo que en el siguiente apartado se analizará la raíz del problema.

Por último, en vista a todos los resultados obtenidos, se puede afirmar que el algoritmo PPO es el más robusto de los cinco. De este modo, se tomará como caso

	A2C	PPO	DDPG	SAC	TD3
Annual return	0,080	0,107	0,108	0,090	0,101
Cumulative returns	0,377	0,524	0,528	0,429	0,491
Annual volatility	0,210	0,211	0,206	0,210	0,210
Sharpe ratio	0,472	0,586	0,600	0,515	0,564
Calmar ratio	0,244	0,321	0,334	0,278	0,313
Stability	0,752	0,832	0,860	0,825	0,838
Max drawdown	−0,328	−0,333	−0,322	−0,323	−0,322
Omega ratio	1,102	1,130	1,133	1,112	1,123
Sortino ratio	0,658	0,819	0,842	0,719	0,786
Skew	−0,368	−0,368	−0,259	−0,300	−0,331
Kurtosis	17,266	18,236	18,397	17,262	16,926
Tail ratio	0,854	0,864	0,842	0,861	0,868
Daily value at risk	−0,026	−0,026	−0,025	−0,026	−0,026

Tabla 5.8. *Estadísticas del Porfolio para todos DRL- Rebalanceo trimestral*

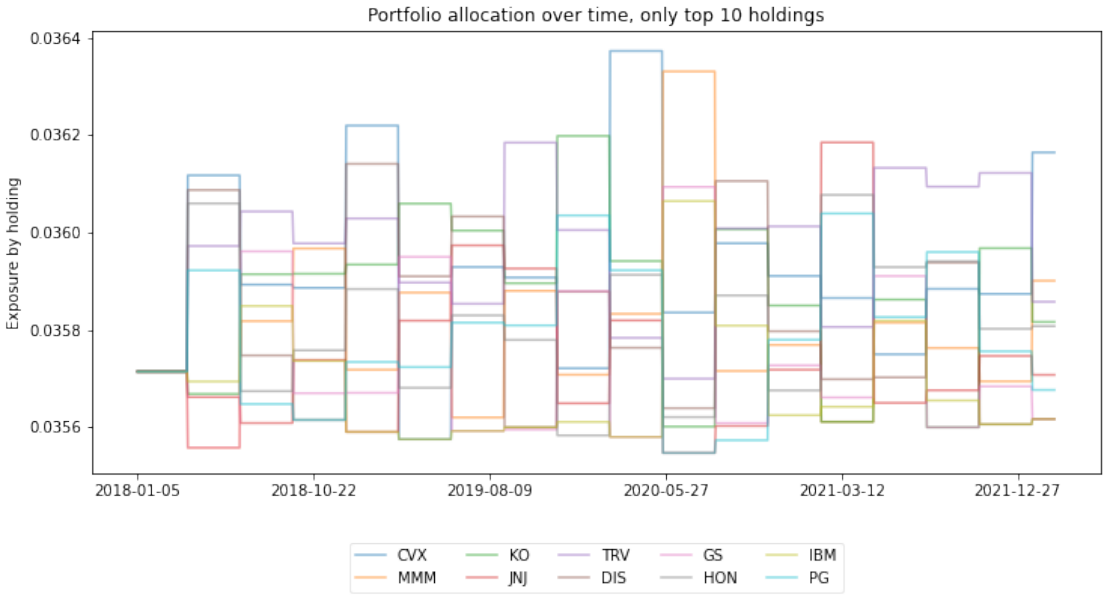


Figura 5.12. *Evolución de la asignación de capital para el rebalanceo trimestral*

base para futuras comparaciones los rendimientos obtenidos con este algoritmo en un rebalanceo mensual.

5.2. Escenario II: función de recompensa e inicialización de los pesos

A continuación, se adjuntan en la Tabla 5.9 los resultados de lanzar 3 simulaciones adicionales que pretenden desafiar al caso base. Se trata de valorar si un cambio en la formulación de la función objetivo, acompañado de una variación en la inicialización de los pesos, puede ofrecer mejores rendimientos.

	Base case	**Markowitz and portfolio value**	**Markowitz and Sharpe ratio**	**Uniform and Sharpe ratio**
Annual return	0,105	−0,007	−0,010	0,102
Cumulative returns	0,516	−0,030	−0,041	0,499
Annual volatility	0,211	0,279	0,280	0,211
Sharpe ratio	0,580	0,134	0,125	0,568
Calmar ratio	0,316	−0,014	−0,020	0,307
Stability	0,830	0,626	0,614	0,821
Max drawdown	−0,333	−0,507	−0,511	−0,333
Omega ratio	1,128	1,036	1,033	1,125
Sortino ratio	0,810	0,159	0,148	0,792
Skew	−0,367	−9,057	−9,119	−0,369
Kurtosis	18,243	192,102	193,950	18,207
Tail ratio	0,862	0,883	0,882	0,862
Daily value at risk	−0,026	−0,035	−0,035	−0,026

Tabla 5.9. *Estadísticas del Porfolio para los distintos modelos PPO*

Como se puede observar en vista a los resultados adjuntos, el hecho de inicializar los pesos con Markowitz no mejora la situación respecto al caso base. Si se analiza de forma más detenida los movimientos que realiza el agente se puede

observar un fenómeno peculiar, representado en la Figura 5.13. Al principio del periodo de validación los pesos se han asignado conforme a ese criterio, pero sin embargo, al cumplirse el mes y tener que forzar un rebalanceo, se distribuye el capital de forma prácticamente equitativa, y ese salto provoca una inestabilidad en el entrenamiento del agente. Por lo tanto, parece que el problema de que los portfolios generados por los modelos de DRL sean como un *equally weighted*, es independiente del método empleado para inicializar los pesos.

Por otra parte, llama la atención que el hecho de cambiar la función objetivo al ratio de Sharpe, no produzca una mejora en el mismo con respecto al caso base. Esto se puede deber a que dicha función sea más costosa de optimizar y por lo tanto, el algoritmo caiga en óptimos locales. Una solución podría pasar por cambiar el valor de los hiperparámetros del modelo.

Figura 5.13. *Problema ocasionado por la inicialización con Markowitz*

5.3. Escenario III: impacto de emplear el autoencoder LSTM

A continuación, se adjunta en la Tabla 5.10 una comparativa de las métricas del portfolio del caso base y del generado mediante la ampliación de los indicadores técnicos con el procedimiento ya descrito. Tal y como se puede observar, el método aumenta ligeramente el rendimiento en conjunto del modelo, por lo que se podría llegar a afirmar que la inclusión de más indicadores técnicos y su posterior reducción con un autoencoder LSTM mejora la eficiencia de la gestión de una cartera de valores.

	Base case	Expanded Indicators
Annual return	0,105	0,108
Cumulative returns	0,516	0,528
Annual volatility	0,211	0,211
Sharpe ratio	0,580	0,589
Calmar ratio	0,316	0,323
Stability	0,830	0,834
Max drawdown	−0,333	−0,333
Omega ratio	1,128	1,130
Sortino ratio	0,810	0,823
Skew	−0,367	−0,370
Kurtosis	18,243	18,250
Tail ratio	0,862	0,862
Daily value at risk	−0,026	−0,026

Tabla 5.10. *Estadísticas del Porftolio para el análisis de los indicadores*

5.4. Escenario IV: filtrado de las acciones según su volatilidad

El objetivo de esta sección es valorar si resultaría conveniente emplear dos modelos de RL según la volatilidad del conjunto de acciones de la cartera. El hecho de ponderar las inversiones según la volatilidad del activo subyacente, puede llevar a mejoras interesantes en el rendimiento de una estrategia (**Moreira**). El razonamiento que subyace a esta idea es que si se discrimina según la variabilidad que presenta la misma quizás se puedan explotar mejor los entornos de forma que el algoritmo pueda alcanzar el óptimo global. De confirmarse esta hipótesis, lo ideal sería construir un bloque anterior a este que prediga la volatilidad del conjunto de acciones a analizar y según su valor, se utilice un modelo u otro.

Para analizar este fenómeno se ha empleado la volatilidad histórica anualizada del conjunto de valores que conforma el índice y se han separado las acciones formando un subconjunto de 10 valores por grupo en cada caso. En el gráfico que se incluye en la Figura 5.14 se representa la contribución total de cada uno de los valores al portfolio, que se define como los rendimientos acumulados por cada uno de los valores a lo largo del tiempo simulado. Por lo tanto, es de esperar que se maximicen dichos porcentajes al aplicar un algoritmo de RL específico para cada entorno.

En la Figuras 5.15 y 5.16 se muestra el mismo gráfico pero computando los porcentajes después de haber aplicado un modelo particularizado según el caso.

Tal y como se puede observar, en las Figuras 5.17 y 5.18 al diseñar un modelo de RL más específico se consiguen mejores rendimientos para las acciones que lo componen, en la mayoría de los casos.

Por último, con respecto a indicadores propios de la gestión de una cartera, resulta interesante comparar el comportamiento del ratio de Sharpe móvil y de la volatilidad propia del portfolio en ambos escenarios y tomando por referencia la actuación del índice. Tal y como se puede observar en la Figura 5.19 el ratio de Sharpe móvil de la cartera formada por el conjunto de acciones del índice menos volátil supera al del índice en prácticamente todo el histórico. Por contra, el portfolio formado por las acciones más volátiles presenta un ratio de Sharpe móvil muy similar al índice, y esto se debe a que, aunque la volatilidad de dicha cartera es mayor, también lo son proporcionalmente sus beneficios esperados. Por otra par-

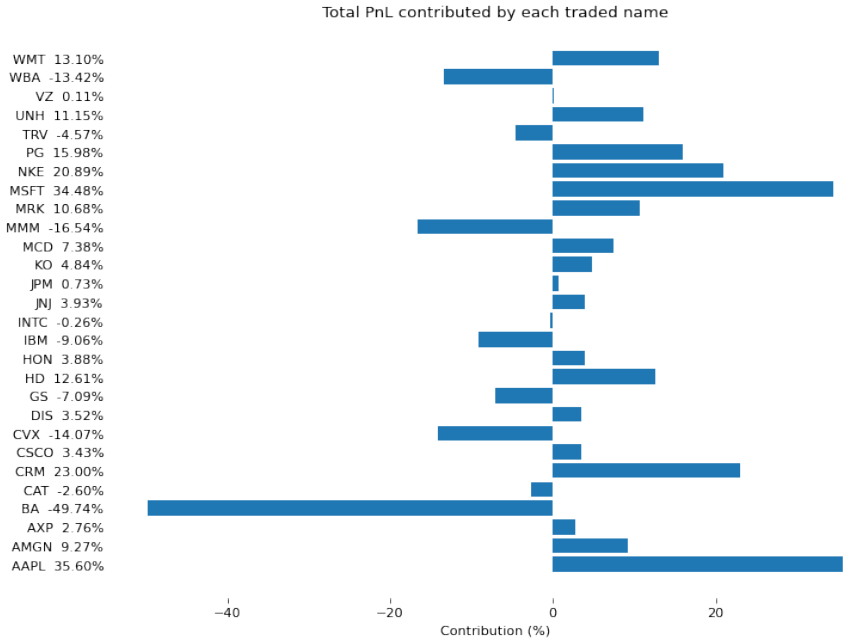


Figura 5.14. *Contribución por acción en el caso base*

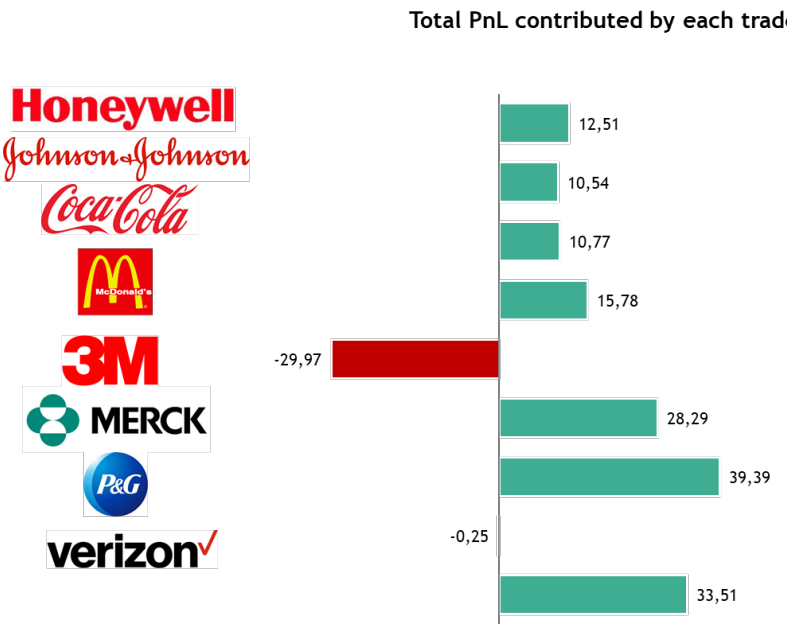


Figura 5.15. *Contribución por acción en el caso base*

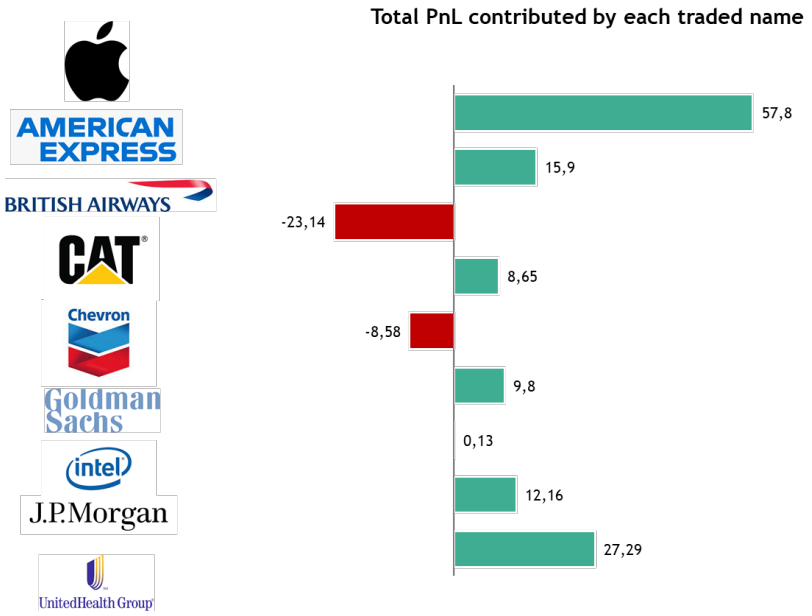


Figura 5.16. *Contribución por acción en el caso base*

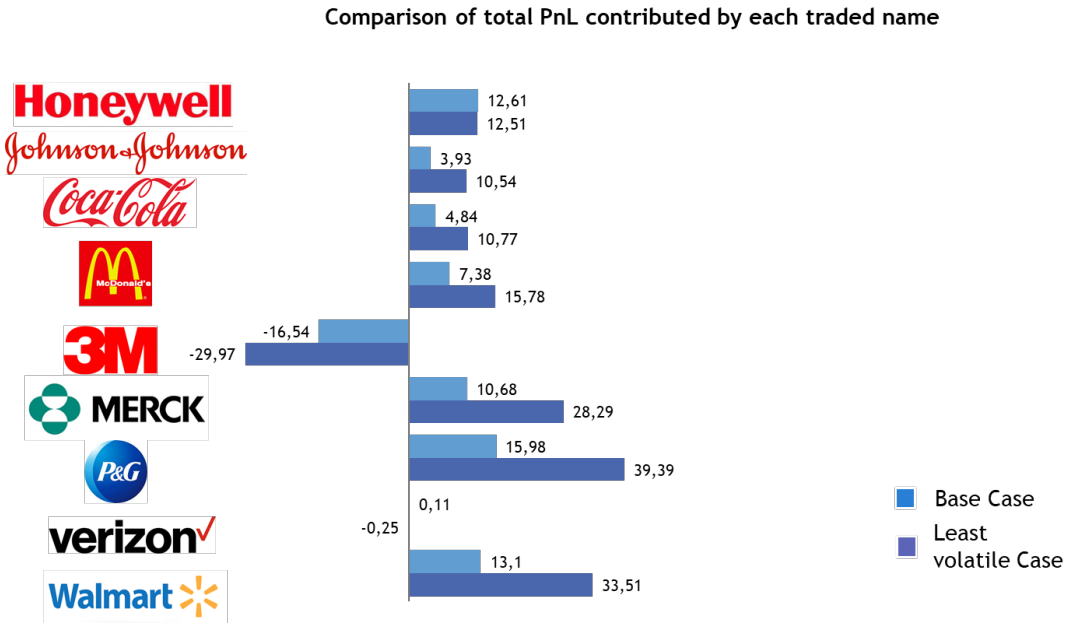


Figura 5.17. *Contribución por acción en el caso base*

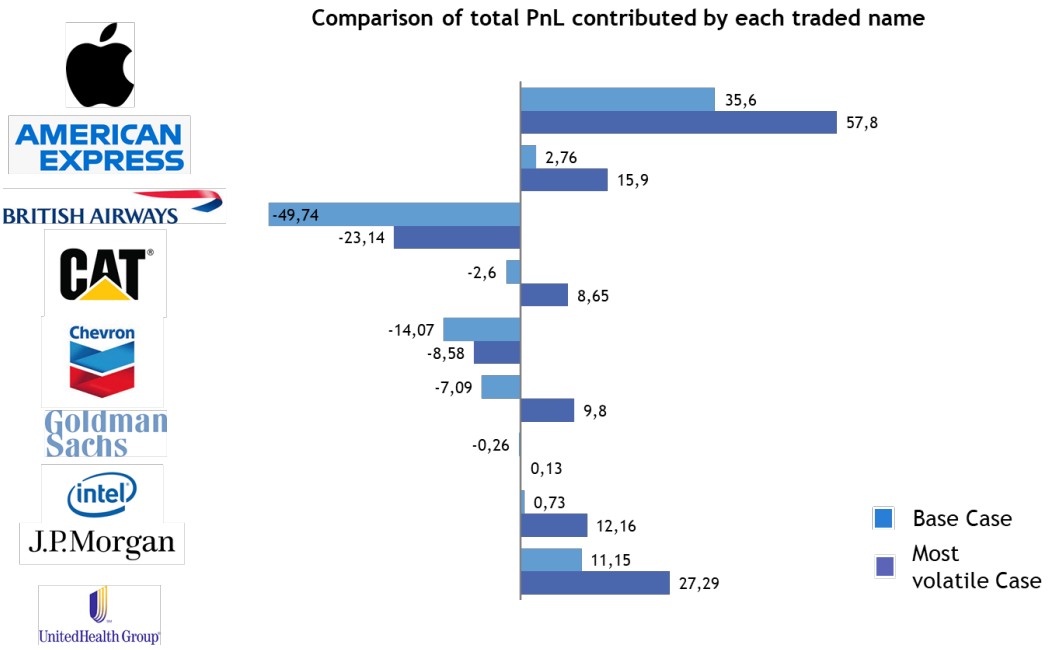
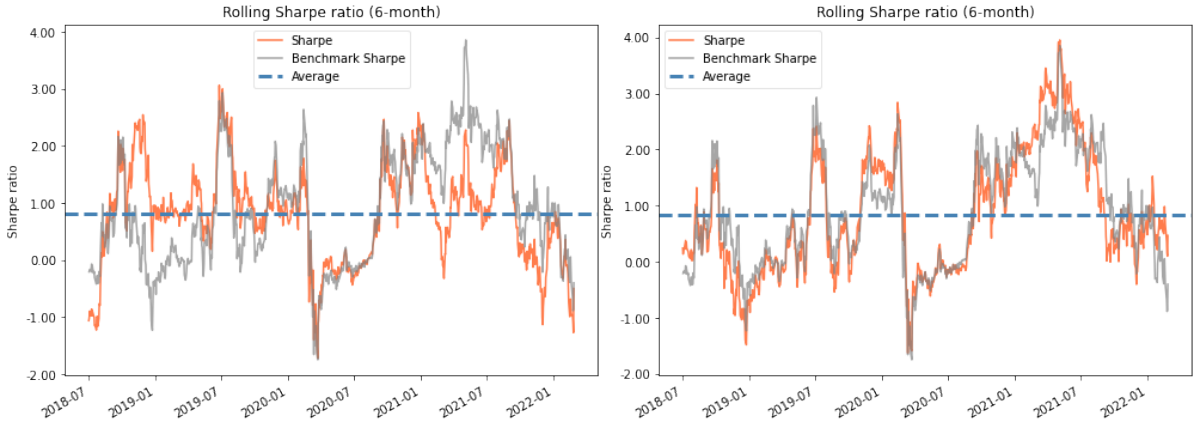


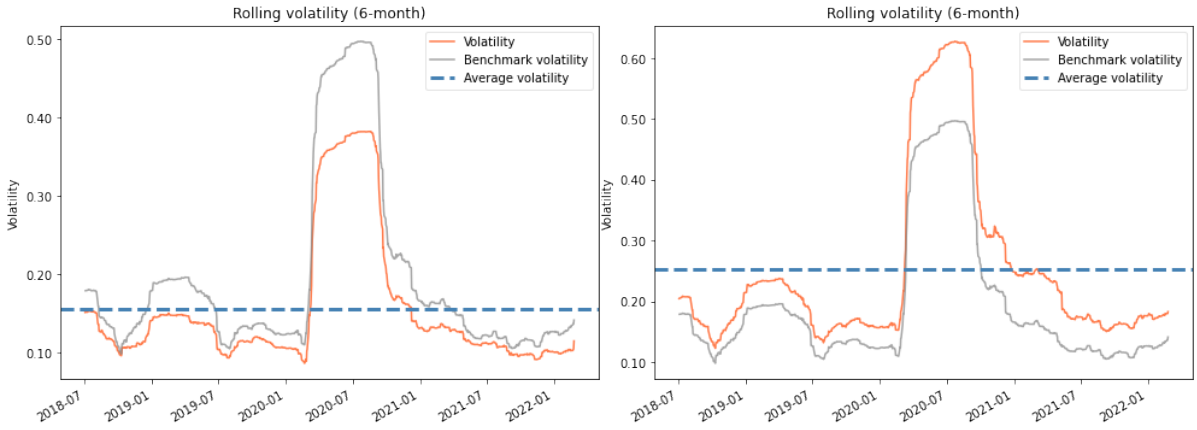
Figura 5.18. *Contribución por acción en el caso base*

te, con respecto a la volatilidad móvil del portfolio, se puede visualizar como en la Figura 5.20 se obtiene la respuesta esperada: la cartera más volátil tiene una volatilidad mayor que el índice y lo opuesto le ocurre a la menos volátil.



(a) *Menos volátil* **(b)** *Más volátil*

Figura 5.19. *Ratio de Sharpe móvil del portfolio*



(a) *Menos volátil* **(b)** *Más volátil*

Figura 5.20. *Volatilidad móvil del portfolio*

5.5. Escenario V: influencia del sector

En este apartado se pretende analizar si hay algún sector que se ha visto más favorecido por el algoritmo. Para ello, en la Figura 5.21 se adjunta la evolución de la contribución de cada sector sobre la asignación total de la cartera. Como las diferencias entre ellos son muy significativas, parece que cada contribución es uniforme, cuando realmente se debe a un problema de escala local. Con este gráfico se confirma que el sector al que el agente ha destinado mayor capital es el tecnológico.

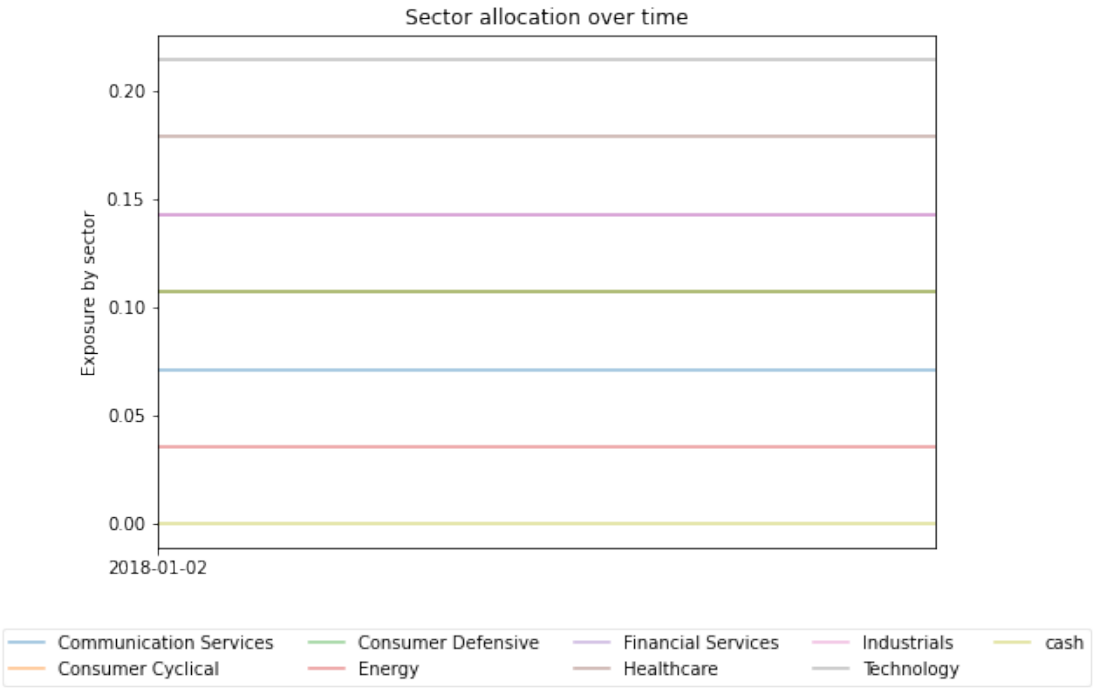


Figura 5.21. *Evolución de la asignación de capital por sectores*

5.6. Escenario VI: validación cruzada del modelo en distintos instantes temporales

La principal motivación de llevar a cabo este análisis es que hay algoritmos que se comportan relativamente bien frente a un conjunto muy específico de condiciones que se dan de forma simultánea, pero su rendimiento no es generalizable a otros entornos. Para ello, se ha dividido el conjunto total de datos en tres particiones de igual duración (adjuntos en la Tabla 5.11) y se ha analizado el comportamiento del modelo *out-of-sample* frente a diferentes situaciones del mercado. En todos los escenarios, se compara el rendimiento del algoritmo con la estrategia de comprar y mantener el índice.

Partición	**Periodo entrenamiento**	**Periodo validación**
1	2008-01-01 – 2011-01-01	2011-01-001 – 2015-01-01
2	2011-01-001 – 2015-01-01	2015-01-01 – 2019-01-01
3	2015-01-01 – 2019-01-01	2019-01-01 – 2022-02-28

Tabla 5.11. *Particiones para los distintos K-Folds*

En todos los escenarios el modelo presentado supera al índice, siendo pues un método robusto. Por otra parte, resulta muy interesante estudiar el comportamiento de los retornos mensuales en cada uno de los escenarios anteriores. Para ello, se adjuntan las Figuras 5.22, 5.23 y 5.24. Tal y como se puede observar, los rendimientos no están concentrados en ningún mes específico, por lo que, el producto de la rentabilidad no proviene de unas condiciones determinadas (e.g., estacionalidad), sino que se genera de forma homogénea en el tiempo.

	PPO model	DJI
Annual return	0,156	0,115
Cumulative returns	0,786	0,539
Annual volatility	0,143	0,142
Sharpe ratio	1,085	0,837
Calmar ratio	0,996	0,682
Stability	0,951	0,913
Max drawdown	−0,157	−0,168
Omega ratio	1,218	1,163
Sortino ratio	1,549	1,179
Skew	−0,395	*NaN*
Kurtosis	4,936	*NaN*
Tail ratio	0,938	0,950
Daily value at risk	−0,017	−0,017

Tabla 5.12. *Estadísticas del Porfolio para el primer K-Fold*

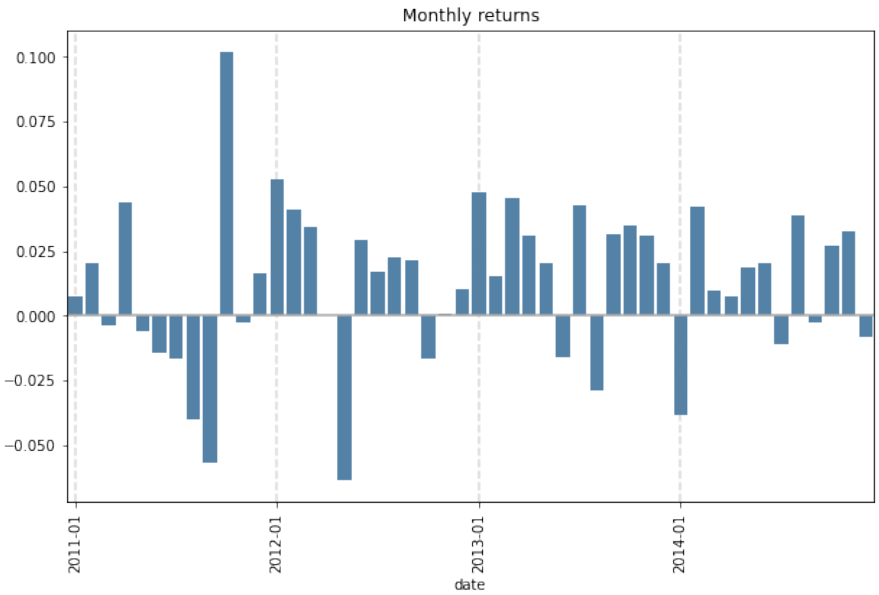


Figura 5.22. *Distribución mensual de los rendimientos para el primer K-Fold*

	PPO model	DJI
Annual return	0,088	0,065
Cumulative returns	0,399	0,282
Annual volatility	0,136	0,135
Sharpe ratio	0,686	0,532
Calmar ratio	0,555	0,437
Stability	0,897	0,860
Max drawdown	−0,158	−0,148
Omega ratio	1,135	1,100
Sortino ratio	0,955	0,726
Skew	−0,376	NaN
Kurtosis	3,973	NaN
Tail ratio	0,930	0,941
Daily value at risk	−0,017	−0,017

Tabla 5.13. *Estadísticas del Porfolio para el segundo K-Fold*

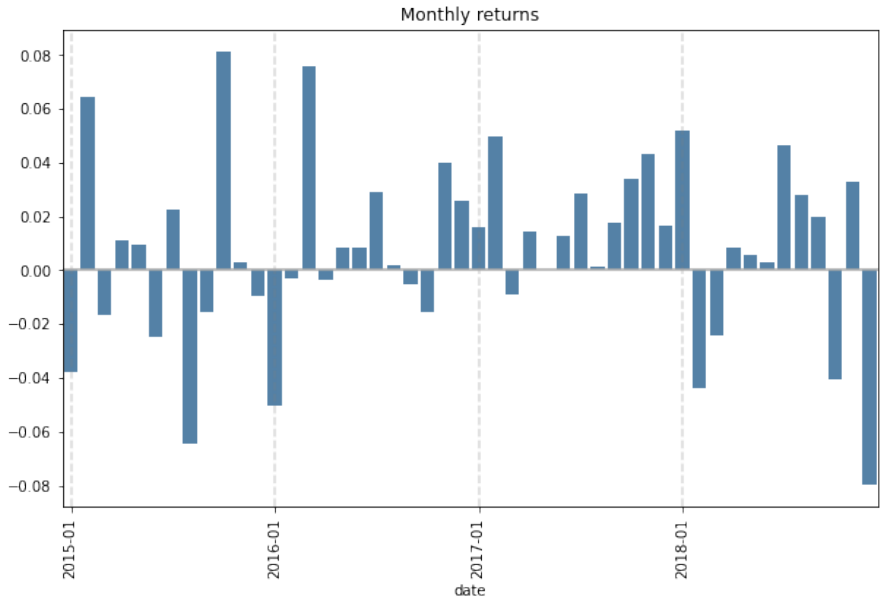


Figura 5.23. *Distribución mensual de los rendimientos para el segundo K-Fold*

	PPO model	DJI
Annual return	0,144	0,140
Cumulative returns	0,527	0,504
Annual volatility	0,223	0,233
Sharpe ratio	0,714	0,684
Calmar ratio	0,431	0,380
Stability	0,795	0,701
Max drawdown	−0,333	−0,371
Omega ratio	1,1166	1,160
Sortino ratio	1,007	0,945
Skew	−0,365	NaN
Kurtosis	19,029	NaN
Tail ratio	0,928	0,939
Daily value at risk	−0,027	−0,029

Tabla 5.14. *Estadísticas del Porfolio para el tercer K-Fold*

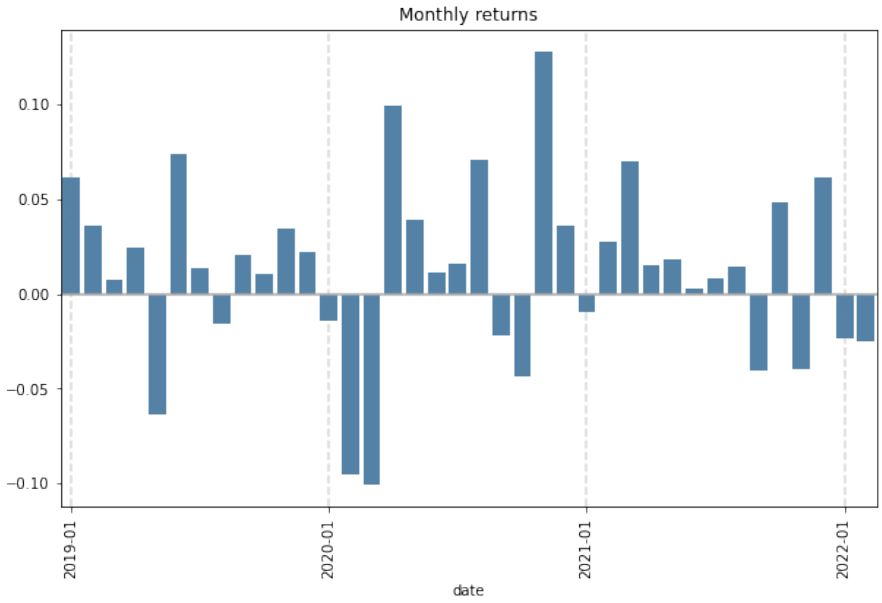


Figura 5.24. *Distribución mensual de los rendimientos para el tercer K-Fold*

5.7. Escenario VII: comparación del modelo de RL con los benchmarks

Finalmente para acabar con el capítulo de la presentación de los resultados, se compara la mejor estrategia encontrada hasta el momento, con el comportamiento de distintos benchmarks. En primer lugar, se comienza la sección contrastando el rendimiento del algoritmo de DRL y de la estrategia de comprar y mantener el índice. En la Figura 5.25 se comparan los retornos acumulados de ambas estrategias. Tal y como se puede observar, el modelo supera al índice a lo largo de todo el periodo analizado.

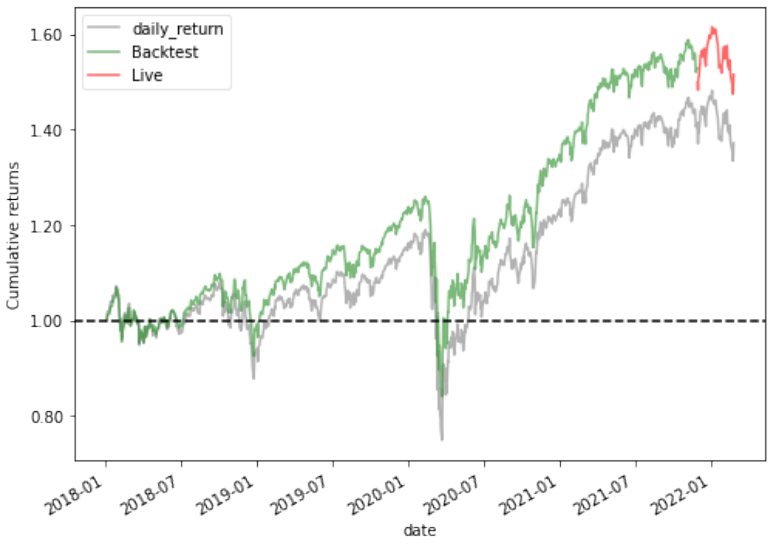


Figura 5.25. *Retornos acumulados*

A continuación, en las Figuras 5.26, 5.27 y 5.28 gráficos que muestran la evolución de la β, el ratio de Sharpe y la volatilidad móvil de la cartera frente al índice. Los resultados son muy parecidos en todas las situaciones debido al subconjunto de acciones empleados.

Por último, en la Figura 5.29 se muestra un gráfico que representa la distribución de los rendimientos anuales. En general los beneficios anuales son muy satisfactorios, teniendo en cuenta los entornos tan difíciles que el mercado ha vivido a lo largo de estos últimos dos años.

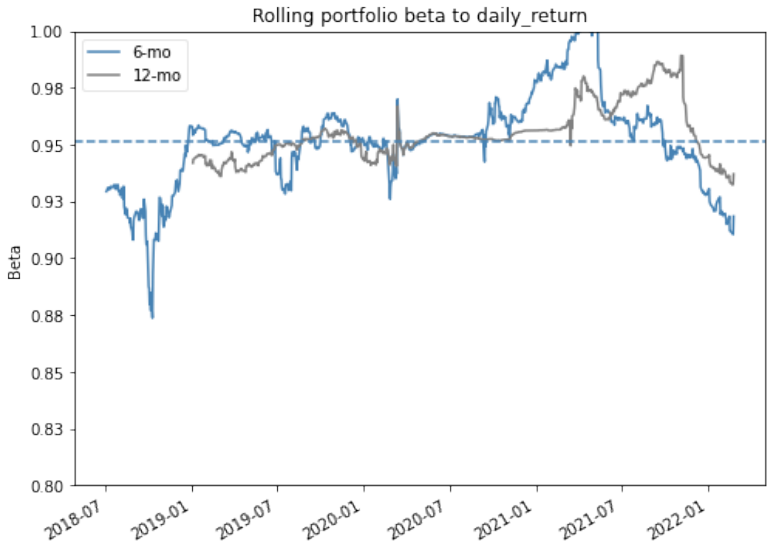


Figura 5.26. *Beta móvil del portfolio*

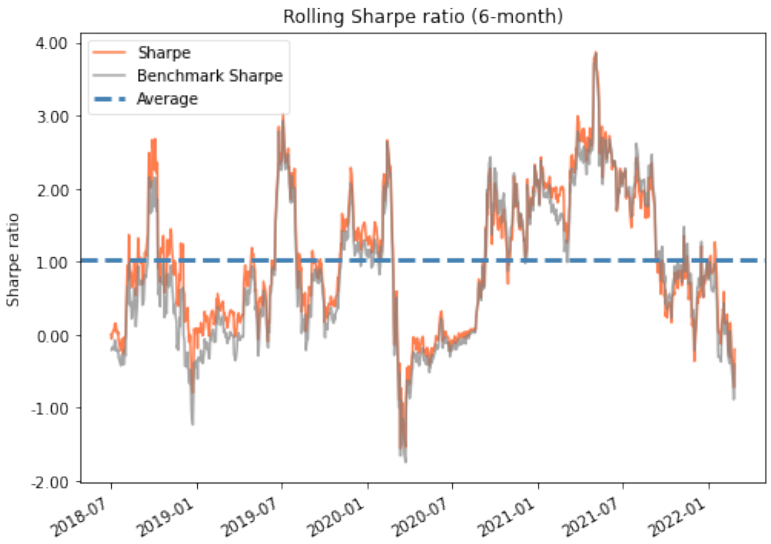


Figura 5.27. *Ratio de Sharpe móvil*

Sin embargo, el principal problema de comparar la estrategia diseñada con el comportamiento del índice es que este último no es negociable en los mercados, de aquí surge la necesidad de diseñar un marco comparativo más amplio. Para ello,

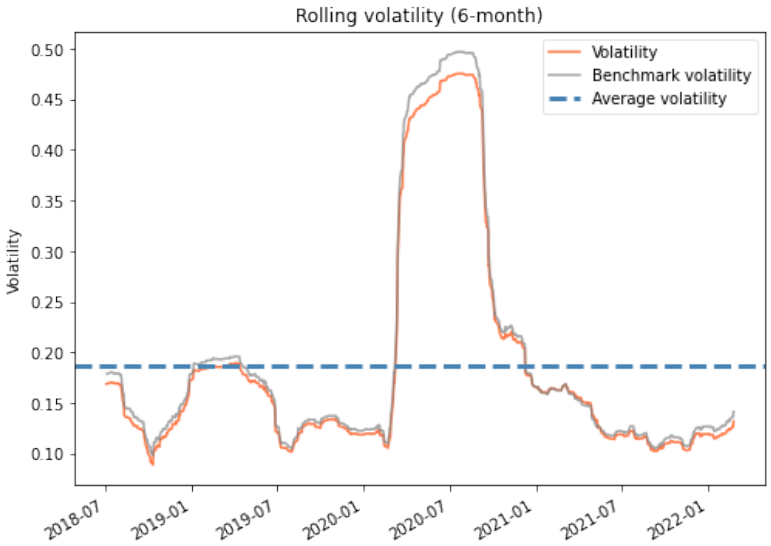


Figura 5.28. *Volatilidad móvil del portfolio*

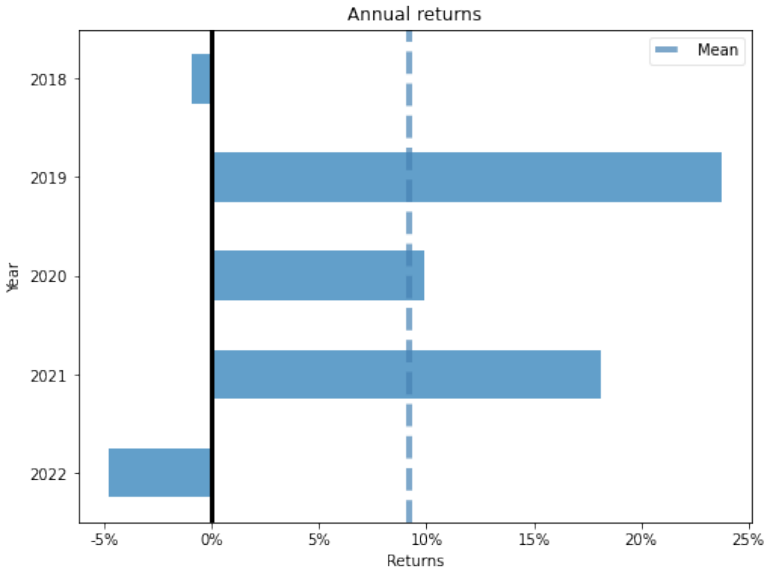


Figura 5.29. *Distribución de los rendimientos anuales*

se enfrenta el rendimiento del algoritmo con las tres estrategias siguientes:

1. Estrategia de comprar y mantener un ETF del índice, concretamente el *SPDR Dow Jones Industrial Average ETF*

2. Estrategia basada en la Teoría Moderna del Portfolio (MPT), con el objetivo de evaluar cómo se pueden construir carteras de activos para optimizar el retorno esperado en función del nivel de riesgo dispuesto a asumir. Para ello, la diversidad de la asignación del capital es un componentes clave. En este escenario, el rebalanceo se produce con la misma frecuencia que en el caso del algoritmo de DRL y asumiendo los mismos costes de transacción, para que la comparación sea lo más fidedigna posible

3. Estrategia de un portoflio equiponderado, en el que la ponderación de cada acción tiene el mismo peso y, por lo tanto, no se rebalancea el portoflio en el periodo examinado.

El resultado de la comparación de dichas estrategias se incluye en la Figura 5.30. Resulta muy interesante cómo el rendimiento del algoritmo es muy similar al de un portfolio *equally-weighted*, y que estas estrategias superan considerablemente a las otras dos. El hecho de que la estrategia de inversión guarde una estrecha relación con la primera mencionada, ha de ser estudiado con más detalle para entender el funcionamiento del método. Posiblemente, si se optimizarán los hiperparámetros de los modelos de DRL se llegarían a resultados sustancialmente diferentes.

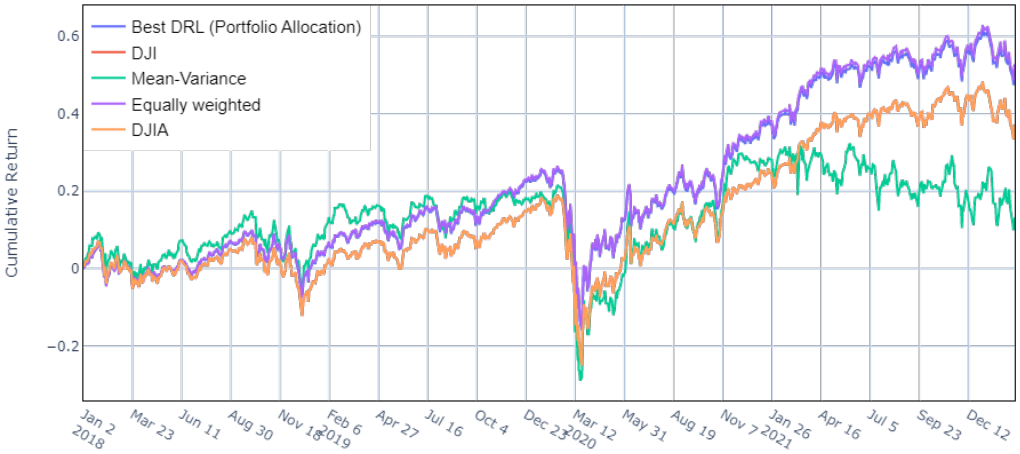


Figura 5.30. *Comparación del modelo de RL con los benchmarks*

Capítulo 6

Conclusiones y futuros desarrollos

En el presente trabajo se ha puesto de manifiesto cómo el algoritmo diseñado es capaz de batir al mercado de referencia de forma recurrente. Por lo tanto, se ha demostrado que las técnicas derivadas del aprendizaje por refuerzo permiten aumentar los beneficios de las operaciones realizadas al optimizar los pesos de una cartera.

Las simulaciones realizadas permiten vislumbrar el impacto de las comisiones en la rentabilidad final del modelo. De este modo, para reducir al mínimo la fricción entre las investigaciones teóricas y la práctica real, se ha diseñado un algoritmo robusto que se adapte a las distintas situaciones del mercado, permitiendo ajustar el periodo de rebalanceo, la volatilidad de la cartera y el empleo de indicadores técnicos, entre otros. Es precisamente la decisión sobre qué entradas utilizar para entrenar el modelo la que se deja para futuros desarrollos, dejando la puerta abierta a incluir variables más relacionadas con el análisis fundamental y macroeconómico.

Apéndice A

Regularización de la entropía

La entropía indica, en términos brutos, cómo de aleatoria una variable es. Por ejemplo, si una moneda esta trucada de forma que la mayoría de las veces que se lanza sale cara, entonces tiene poca entropía; si por el contrario, no lo está y es equiprobable que salga cara o cruz, entonces posee una elevada entropía.

Si se define x como una variable aleatoria con una función de probabilidad P. La entropía H DE x se puede definir como:

$$H(P) = \mathrm{E}_{x\sim P}[-\log P(x)] \tag{A.1}$$

En el ámbito del aprendizaje por refuerzo, el agente obtiene un bono adicional en cada paso proporcional al valor de la entropía de la política en cada timestep. De este modo, esto da lugar a una transformación del problema según la siguiente ecuación:

$$\pi^* = \arg\max_{\pi} \mathrm{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \left(R\left(s_t, a_t, s_{t+1}\right) + \alpha H\left(\pi\left(\cdot \mid s_t\right)\right)\right)\right] \tag{A.2}$$

donde $\alpha > 0$ es el coeficiente de *trade-off* entre la recompensa y la entropía. Como consecuencia, se puede redefinir $V^\pi(s)$ como sigue:

$$V^\pi(s) = \mathrm{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t \left(R\left(s_t, a_t, s_{t+1}\right) + \alpha H\left(\pi\left(\cdot \mid s_t\right)\right)\right) \mid s_0 = s\right] \tag{A.3}$$

donde Q^π se cambia para incluir el término de la entropía en cada *timestep* salvo el primero

$$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{\mathrm{E}}\left[\sum_{t=0}^{\infty}\gamma^{t}R\left(s_{t},a_{t},s_{t+1}\right)+\alpha\sum_{t=1}^{\infty}\gamma^{t}H\left(\pi\left(\cdot\mid s_{t}\right)\right)\mid s_{0}=s,a_{0}=a\right] \tag{A.4}$$

Con estas definiciones, V^{π} y Q^{π} están conectados mediante la siguiente relación:

$$V^{\pi}(s) = \underset{a\sim\pi}{\mathrm{E}}\left[Q^{\pi}(s,a)\right]+\alpha H(\pi(\cdot\mid s)) \tag{A.5}$$

De donde se puede derivar la ecuación de Bellman para Q^{π} :

$$\begin{aligned} Q^{\pi}(s,a) &= \underset{\substack{s'\sim P\\ a'\sim\pi}}{\mathrm{E}}\left[R\left(s,a,s'\right)+\gamma\left(Q^{\pi}\left(s',a'\right)+\alpha H\left(\pi\left(\cdot\mid s'\right)\right)\right)\right] \\ &= \underset{s'\sim P}{\mathrm{E}}\left[R\left(s,a,s'\right)+\gamma V^{\pi}\left(s'\right)\right]. \end{aligned} \tag{A.6}$$

Apéndice B

Explicación adicional de las partes del DDPG

En este apéndice se explica de forma exhaustiva las dos partes del DDPG: el aprendizaje de una función Q y el aprendizaje de la política.

B.1. Parte Q-learning del DDPG

En primer lugar, conviene recapitular la ecuación de Bellman que describe la función acción-valor óptima, $Q^*(s,a)$. Esta viene dada por:

$$Q^*(s,a) = \underset{s' \sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right] \tag{B.1}$$

donde $s' \sim P$ es una simplificación para referir al siguiente estado $s^{'}$ que se samplea del entorno según una distribución $P(\cdot \mid s,a)$.

Esta ecuación de Bellman es el punto de partida para aprender un aproximador a $Q^*(s,a)$. Supóngase que el aproximador es una red neuronal $Q_\phi(s,a)$, con parámetros ϕ, y que se ha recogido un conjunto $\mathcal{D}$ de transiciones $(s,a,r,s^{'},d)$ (donde d indica si el estado $s^{'}$ es terminal). Se puede establecer una función de error medio-cuadrado de Bellman (*MSBE*), que indica aproximadamente lo cerca que se encuentra Q_ϕ de satisfacer la ecuación de Bellman:

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d) \sim \mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) \max_{a'} Q_\phi(s',a') \right) \right)^2 \right] \tag{B.2}$$

Los algoritmos de Q-learning para aproximadores de funciones, como DQN (y todas sus variantes) y DDPG, se basan en gran medida en la minimización de esta función de pérdida *MSBE*. Hay dos ajustes principales empleados por todos ellos que vale la pena describir, y luego un detalle específico para DDPG.

B.1.1. Primer ajuste: Búfer de repetición

Todos los algoritmos estándar para el entrenamiento de una red neuronal profunda para aproximar $Q^*(s, a)$ hacen uso de un búfer de repetición de experiencias. Este es el conjunto $\mathcal{D}$ de experiencias anteriores. Para que el algoritmo tenga un comportamiento estable, el búfer de repetición debe ser lo suficientemente grande como para contener una amplia gama de experiencias, pero no siempre es bueno mantener todo. Si sólo utiliza los datos más recientes, se ajustará demasiado a ellos; si utiliza demasiada experiencia, puede ralentizar el aprendizaje.

B.1.2. Segundo ajuste: Redes objetivo

Los algoritmos de Q-learning utilizan redes objetivo. El término:

$$r + \gamma(1 - d) \max_{a'} Q_\phi (s', a') \tag{B.3}$$

se llama el objetivo, porque cuando se minimiza la función de pérdida *MSBE*, se está tratando de hacer que la función Q sea más parecida a este objetivo. El problema es que el objetivo depende de los mismos parámetros que estamos se están entrenando: ϕ. Esto hace que la minimización *MSBE* sea inestable. La solución es utilizar un conjunto de parámetros que se acerque a ϕ, pero con un retardo de tiempo, es decir, una segunda red, llamada red objetivo, que va por detrás de la primera. Los parámetros de la red objetivo se denominan ϕ_{targ}.

En los algoritmos basados en DQN, la red objetivo se copia de la red principal cada cierto número de pasos. En los algoritmos de estilo DDPG, la red objetivo se actualiza una vez por cada actualización de la red principal mediante el promedio de *polyak*:

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi \tag{B.4}$$

donde ρ es un hiperparámetro entre 0 y 1 (frecuentemente más cercano a 1).

B.1.3. Tercer ajuste: detalle del DDPG

La principal modificación es el cálculo del máximo sobre acciones en el objetivo. Como se mencionó anteriormente calcular el máximo sobre las acciones en el objetivo es un desafío en los espacios de acción continua. El DDPG se ocupa de esto mediante el uso de una red de política de destino para calcular una acción que aproximadamente maximiza $Q_{\phi_{targ}}$. La red política objetivo se encuentra de la misma manera que la función Q objetivo: promediando los parámetros de la política a lo largo del entrenamiento. En conjunto, el aprendizaje de Q en DDPG se realiza minimizando la siguiente pérdida *MSBE* con el descenso de gradiente estocástico:

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) Q_{\phi_{\text{targ}}} \left(s', \mu_{\theta_{\text{targ}}} \left(s' \right) \right) \right) \right)^2 \right] \quad \text{(B.5)}$$

B.2. Parte Q-learning del DDPG

El aprendizaje de políticas en DDPG es bastante sencillo. Se pretende aprender una política determinista $\mu_\theta(s)$ que da la acción que maximiza $Q_\phi(s,a)$. Dado que el espacio de acción es continuo, y se asume que la función Q es diferenciable con respecto a la acción, se puede, simplemente, realizar el ascenso de gradiente (con respecto a los parámetros de la política solamente) para resolver:

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}} \left[Q_\phi \left(s, \mu_\theta(s) \right) \right] \quad \text{(B.6)}$$

Apéndice C

Estadísticas del portfolio

- **Ratio de Calmar**

  El ratio de Calmar es un parámetro del rendimiento entre la rentabilidad anualizada y el máximo drawdown del activo en cuestión. Se computa teniendo en cuenta los datos de los últimos 36 meses. El drawdown se puede definir como la diferencia entre el máximo de referencia de una curva y el mínimo, en un período de tiempo. Es una medida de riesgo que se utiliza en el ratio de Calmar, en lugar de la volatilidad. Se computa de acuerdo a la siguiente relación:

$$\text{Ratio de Calmar } = \frac{\text{Rentabilidad anualizada}}{\mid \text{ Máximo drawdown } \mid} \tag{C.1}$$

- **Ratio de Sharpe**

  Este ratio financiero fue desarrollado por el nobel de economía William F. Sharpe para saber si la rentabilidad de una inversión se debe a una decisión inteligente o, si por el contrario, es resultado de haber asumido más riesgo. Es un ratio que calcula la rentabilidad ajustada según su riesgo. El ratio de Sharpe se calcula restando la rentabilidad de un activo sin riesgo (como puede ser la deuda a 10 años de Alemania) a la rentabilidad de la inversión y dividendo el resultado entre el riesgo, calculado como la desviación típica de la rentabilidad de la inversión, tal y como se observa en la siguiente fórmula:

$$S = \frac{R - R_f}{\sigma} \tag{C.2}$$

- **Ratio de Sortino**
  Esta métrica muestra el exceso de rendimiento por encima de un determinado objetivo por unidad de riesgo a la baja.

$$S = \frac{R - T}{DR} \tag{C.3}$$

donde DR es la desviación estándar de los rendimientos negativos (downside risk) representado matemáticamente por:

$$DR = \sqrt{\int_{-\infty}^{t} (T - r)^2 f(r) dr} \tag{C.4}$$

La fórmula del ratio de Sortino, es similar a la del ratio de Sharpe. En su cálculo considera el exceso de rentabilidad esperada ($Rp\breve{}Rf$) sobre el activo sin riesgo; sin embargo, la medida de volatilidad utilizada para corregir el ratio por riesgo no tiene nada que ver con la tradicional desviación estándar (Sp), utilizándose en su lugar, una medida de dispersión de los rendimientos negativos. Este indicador fue desarrollado para diferenciar entre "la buena" y "la mala" volatilidad en el ratio de Sharpe. De esta forma, si en el período una inversión financiera tuvo un extraordinario comportamiento al alza, la mayor volatilidad o desvío estándar que mostrara por esta causa, no sería considerada en la parte del análisis de la rentabilidad que considera el riesgo, medido éste como la probabilidad de que se produzcan pérdidas. Los inversores, de acuerdo con Sortino, no están asustados por las oscilaciones de su inversión en sí misma, sino que se sienten presionados sólo en un entorno de resultados que no lleguen a lo que ellos consideran un mínimo aceptable (en muchos casos, el activo libre de riesgo).

- **Ratio de Omega**

Se define como la relación ponderada por la probabilidad de las ganancias frente a las pérdidas para un objetivo de rentabilidad determinado. El ratio es una alternativa al ratio de Sharpe y se basa en la información que la relación de Sharpe descarta. El Omega se calcula creando una partición en la distribución de la rentabilidad acumulada para crear un área de pérdidas y un área de ganancias en relación con este umbral. Se computa matemáticamente atendiendo a la siguiente fórmula:

$$\Omega(\theta) = \frac{\int_{\theta}^{\infty}[1 - F(r)]dr}{\int_{-\infty}^{\theta} F(r)dr} \tag{C.5}$$

donde F es la distribución de probabilidad acumulada de la función de retornos y θ es un umbral objetivo que establece qué se considera perdida y qué ganancia. Un valor de Omega alto implica que la cartera proporciona más ganancias relativas que pérdidas en relación con el umbral establecido.

- **Ratio de cola**

  Este ratio mide la probabilidad de que una inversión se vaya a alejar más de tres desviaciones estándar de su media de lo que resultaría de una distribución normal.

Bibliografía

Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A. (2017). Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, *34*(6), 26-38. https://doi.org/10.1109/MSP.2017.2743240

Azhikodan, A. R., Bhat, A. G. K. & Jadhav, M. V. (2019). Stock Trading Bot Using Deep Reinforcement Learning.

Barto, A. G., Sutton, R. S. & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-13*(5), 834-846. https://doi.org/10.1109/TSMC.1983.6313077

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, *6*(5), 679-684. http://www.jstor.org/stable/24900506

Chen, L. & Gao, Q. (2019). Application of Deep Reinforcement Learning on Automated Stock Trading. *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, 29-33.

Deng, Y., Bao, F., Kong, Y., Ren, Z. & Dai, Q. (2017). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems*, *28*, 653-664.

Du, X. & Zhai, J. (s.f.). Algorithm Trading using Q-Learning and Recurrent Reinforcement Learning.

Fischer, T. G. (2018). *Reinforcement learning in financial markets - a survey* (FAU Discussion Papers in Economics N.º 12/2018). Nürnberg, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institute for Economics. http://hdl.handle.net/10419/183139

Guo, Y., Fu, X., Shi, Y. & Liu, M. (2018). Robust Log-Optimal Strategy with Reinforcement Learning. *arXiv: Portfolio Management*.

Huang, C. (2018). Financial Trading as a Game: A Deep Reinforcement Learning Approach.

Jiang, Z., Xu, D. & Liang, J. (2017). A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. *ArXiv*, *abs/1706.10059*.

Jin, O. & El-Saawy, H. (2016). Portfolio Management using Reinforcement Learning.

Li, Y. (2017). Deep Reinforcement Learning: An Overview. *CoRR*, *abs/1701.07274*. http://arxiv.org/abs/1701.07274

Liang, Z., Chen, H., Zhu, J., Jiang, K. & Li, Y. (2018). *Adversarial Deep Reinforcement Learning in Portfolio Management* (Papers N.º 1808.09940). arXiv.org. https://ideas.repec.org/p/arx/papers/1808.09940.html

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529-533.

Mohan, V., Singh, J. G. & Ongsakul, W. (2016). Sortino Ratio Based Portfolio Optimization Considering EVs and Renewable Energy in Microgrid Power Market. *IEEE Transactions on Sustainable Energy*, *8*, 1-1. https://doi.org/10.1109/TSTE.2016.2593713

Nabipour, M., Nayyeri, P., Jabani, H., Band, S. & Mosavi, A. (2020). Predicting stock market trends using machine learning and deep learning algorithms via continuous and binary data; a comparative analysis on the Tehran stock exchange. *IEEE Access*, *PP*, 1-1. https://doi.org/10.1109/ACCESS.2020.3015966

Norris, J. R. (1998). *Markov chaings*. Cambridge university Press.

Ozbayoglu, A. M., Gudelek, M. U. & Sezer, O. B. (2020). Deep learning for financial applications : A survey. *Applied Soft Computing*, *93*, 106384. https://doi.org/https://doi.org/10.1016/j.asoc.2020.106384

Pal, N. & Bezdek, J. (1984). Measuring Fuzzy uncertainty. *IEEE*, *2*, 107-118.

Soleymani, F. & Paquet, E. (2020). Financial Portfolio Optimization with Online Deep Reinforcement Learning and Restricted Stacked Autoencoder - DeepBreath. *Expert Systems with Applications*, *156*, 113456. https://doi.org/10.1016/j.eswa.2020.113456

Sutton, R. S. & G, A. (2017). *Reinforcement Learning: An Introduction (2nd Edition, in progress)*. MIT Press.

Thakkar, A. & Chaudhari, K. (2021). A comprehensive survey on deep neural networks for stock market: The need, challenges, and future directions. *Expert Systems with Applications*, *177*, 114800. https://doi.org/https://doi.org/10.1016/j.eswa.2021.114800

Weijs, L. (2018). Reinforcement learning in Portfolio Management and its interpretation.

Wu, J., WANG, C., XIONG, L. & SUN, H. (2019). Quantitative Trading on Stock Market Based on Deep Reinforcement Learning, 1-8. https://doi.org/10.1109/IJCNN.2019.8851831

Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V. & Fujita, H. (2020). Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, *538*, 142-158. https://doi.org/https://doi.org/10.1016/j.ins.2020.05.066

Xiong, Z., Liu, X.-Y., Zhong, S., Yang, H. & Walid, A. (2018). *Practical Deep Reinforcement Learning Approach for Stock Trading* (Papers N.º 1811.07522). arXiv.org. https://ideas.repec.org/p/arx/papers/1811.07522.html

Yang, H., Liu, X.-Y., Zhong, S. & Walid, A. (2020). Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy.

Zhang, Z., Zohren, S. & Roberts, S. (2020). Deep Reinforcement Learning for Trading. *The Journal of Financial Data Science*, *2*(2), 25-40. https://doi.org/10.3905/jfds.2020.1.030